Universidade do Minho
Escola de Engenharia

André de Matos Pedro

Dynamic contracts for verification and enforcement of real-time systems properties

**Programa de Doutoramento em Informática (MAP-i)
das Universidades do Minho, de Aveiro e do Porto**

Universidade do Minho

universidade de aveiro

U.PORTO

maio de 2018

André de Matos Pedro **Dynamic contracts for verification and enforcement of real-time systems properties**

UMinho | 2018

Universidade do Minho
Escola de Engenharia

André de Matos Pedro

Dynamic contracts for verification and enforcement of real-time systems properties

Programa de Doutoramento em Informática (MAP-i)
das Universidades do Minho, de Aveiro e do Porto

Universidade do Minho

Trabalho realizado sob a orientação do
Professor Doutor Jorge Sousa Pinto
e do
Professor Doutor Luís Miguel Pinho

maio de 2018

STATEMENT OF INTEGRITY

I hereby declare having conducted my thesis with integrity. I confirm that I have not used plagiarism or any form of falsification of results in the process of the thesis elaboration.
I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, 08/05/2018

Full name: André de Matos Pedro

Signature: 

Abstract

Runtime verification is an emerging discipline that investigates methods and tools to enable the verification of program properties during the execution of the application. The goal is to complement static analysis approaches, in particular when static verification leads to the explosion of states. Non-functional properties, such as the ones present in real-time systems are an ideal target for this kind of verification methodology, as are usually out of the range of the power and expressiveness of classic static analyses.

Current real-time embedded systems development frameworks lack support for the verification of properties using explicit time where counting time (i.e., durations) may play an important role in the development process. Temporal logics targeting real-time systems are traditionally undecidable. Based on a restricted fragment of *Metric temporal logic with durations* (MTL-$\int$), we present the proposed synthesis mechanisms 1) for target systems as runtime monitors and 2) for SMT solvers as a way to get, respectively, a verdict at runtime and a schedulability problem to be solved before execution. The later is able to solve partially the schedulability analysis for periodic resource models and fixed priority scheduler algorithms. A domain specific language is also proposed in order to describe such schedulability analysis problems in a more high level way.

Finally, we validate both approaches, the first using empirical scheduling scenarios for uni-multi-processor settings, and the second using the use case of the lightweight autopilot system Px4/Ardupilot widely used for industrial and entertainment purposes. The former also shows that certain classes of real-time scheduling problems can be solved, even though without scaling well. The later shows that for the cases where the former cannot be used, the proposed synthesis technique for monitors is well applicable in a real world scenario such as an embedded autopilot flight stack.

Resumo

A verificação do tempo de execução é uma disciplina emergente que investiga métodos e ferramentas para permitir a verificação de propriedades do programa durante a execução da aplicação. O objetivo é complementar abordagens de análise estática, em particular quando a verificação estática se traduz em explosão de estados. As propriedades não funcionais, como as que estão presentes em sistemas em tempo real, são um alvo ideal para este tipo de metodologia de verificação, como geralmente estão fora do alcance do poder e expressividade das análises estáticas clássicas.

As atuais estruturas de desenvolvimento de sistemas embebidos para tempo real não possuem suporte para a verificação de propriedades usando o tempo explícito onde a contagem de tempo (ou seja, durações) pode desempenhar um papel importante no processo de desenvolvimento. As lógicas temporais que visam sistemas de tempo real são tradicionalmente indecidíveis. Com base num fragmento restrito de MTL-$\int$ (metric temporal logic with durations), apresentaremos os mecanismos de síntese 1) para sistemas alvo como monitores de tempo de execução e 2) para solvers SMT como forma de obter, respectivamente, um veredicto em tempo de execução e um problema de escalonamento para ser resolvido antes da execução. O último é capaz de resolver parcialmente a análise de escalonamento para modelos de recursos periódicos e ainda para algoritmos de escalonamento de prioridade fixa. Propomos também uma linguagem específica de domínio para descrever esses mesmos problemas de análise de escalonamento de forma mais geral e sucinta.

Finalmente, validamos ambas as abordagens, a primeira usando cenários de escalonamento empírico para sistemas uni- multi-processador e a segunda usando o caso de uso do sistema de piloto automático leve Px4/Ardupilot amplamente utilizado para fins industriais e de entretenimento. O primeiro mostra que certas classes de problemas de escalonamento em tempo real podem ser solucionadas, embora não seja escalável. O último mostra que, para os casos em que a primeira opção não possa ser usada, a técnica de síntese proposta para monitores aplica-se num cenário real, como uma pilha de vôo de um piloto automático embebido.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisors Professor Jorge Sousa Pinto and Professor Luís Miguel Pinho, for their guidance, valuable feedback, and encouragement. Without them, this long run could never be finished with success.

My sincere thank you also goes to David Pereira for his encouragement, insightful comments and for being so good advisor and colleague along these years. With him, I have discussed several ideas often useful in this thesis. I want to thank Geoffrey Nelissen for the fruitful meetings and discussions that we had along these years. I also would like to thank Professor Simão Melo de Sousa for pushing me to this wonderful world of the Formal Methods, and also for being an incentive to start my Ph.D. studies at CISTER.

Regarding host institutions, I would like to thank CISTER and Haslab, and University of Minho.

I thank all my CISTER colleagues, Ricardo Severino, Luís Nogueira, Artem Burmyakov, Kostiantyn Berezovskyi, and in special to Claudio Maia and José Fonseca for the opportunity to create a healthy environment to work and enjoy the life with so long mad discussions.

Last but not least, I would like to thank my all family for supporting my humor and my thoughts along these tough years, essentially my parents Eugenia and José, my little brother Eduardo, and my sunshine Sonia for being so cooperative and comprehensive with me.

This thesis was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234); FCOMP-01-0124-FEDER-015006 (VIPCORE) and FCOMP-01-0124-FEDER-020486 (AVIACC); also by FCT and EU ARTEMIS JU, within project ARTEMIS/0003/2012, JU grant nr. 333053 (CONCERTO); and by FCT/MEC and the EU ARTEMIS JU within project ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis considers the field of real-time embedded systems, in which it is crucial to guarantee a correct behavior in the temporal domain [Stankovic, 1988]. These systems range from simple, isolated components to large, highly complex and inherently concurrent systems. They act upon a variety of environments which are frequently very dynamic and hard to capture during design time. Therefore, developing an *real-time system* (RTS) can easily become a very difficult task to complete. Even though RTSs present potentially complex requirements, their design and development processes are mostly limited to model-driven techniques and intensive testing and fault-injection, which are known to allow the existence of human-introduced errors. At later stages of the development cycle such errors can become highly expensive and very hard to tackle, even with the number of static analysis tools available. As the technology evolves, real-time embedded systems are becoming more and more pervasive in our daily routines. Notorious examples of the pervasiveness of real-time embedded systems in our daily lives range from airplane and car control systems to medical devices such as pacemakers. A relevant example which is spurring much interest and that we use in the thesis is the large variety of exciting new models of commercial lightweight multi-copters available in the market and which are currently being intensively used for aerial photography and cinematography, cargo inspection and transportation, and for family entertainment. For safety reasons, some of these multi-copters are being subject to restricted usage rules in several countries to limit their excessively fast spreading in commercial applications. The traditionally adopted mechanisms to treat the failures that can arise during multi-copter activity are commonly applied only for hardware malfunctions. However, in the case of software, the adopted applications/control systems are considerably open for users to modify, which in turn increases the risk for these multi-copters to potentially crash in public areas, namely when several developers spread over the world make changes on these systems. On the more

rigorous side of RTS development, formal methods have been introduced progressively in the development cycle, most of which are based on temporal logic. While standard temporal logics yield a natural and abstract framework for the analysis of safety and liveness properties [Pnueli, 1977], these logics fail to capture the specific timing properties of RTSs [Koymans, 1990]. This limitation is tackled by a set of timed temporal logics [Alur and Henzinger, 1992a], and many of these logics have already been used to develop model checking tools [Behrmann et al., 2006]. However, model checking has its own pitfalls, namely when the size of the state space of the model that captures the RTS under consideration is too large to be mechanically analyzed by a tool implementing a model checking algorithm. Moreover, it might be the case that the properties to be checked cannot be captured rigorously at the abstract level of the model of the system.

When we talk about *Runtime Verification* (RV) of real-time embedded systems, we are increasing the dependability of these systems by drawing verdicts at runtime that may be used to trigger recovery actions. RV is a major complement to static methods as it can be used to check errors for which it is possible to conclude some property of interest based exclusively in knowledge that can be gathered only at execution time. Contrary to *ad hoc* instrumentation of runtime behavior, RV based approaches use formal specifications and synthesize them into *monitors*, that is, pieces of code that take partial traces of execution of the system and match them against the referred specifications and make a verdict. Moreover, monitors can be used both to verify and enforce the properties which are provided by components, even when the components assume the form of a black-box, as long as each component is coupled with a formal specification. A simple example of the power of RV is the case when the response to a property violation detection consists in shutting down a complex component and give control to a simpler, yet formally verified component. By adopting RV techniques, developers can decrease the usual intensive testing efforts, and if used in collaboration with static verification methods, this can increase the overall coverage of the system by ensuring execution time correctness in those parts of the development where heavy-weight static approaches like model checking and deductive verification fail due to well-known problems (*e.g.*, the state-space explosion problem inherent to model checking and the lack of proof automation in deductive verification).

1.1 Problem Statement

In this thesis, we consider the problem of runtime checking hard real-time systems by generating correct-by-construction monitors from a formal language and their correct integration in target applications/systems. The outcome of a monitor checking is a “yes”,

"no" or "unknown" answer. In the case of a gas burner, for example, we may check that the solenoid never leak for more than 4 time units in a period of at most 30 time units. For the case of the system integration, we want to ensure that the monitoring interference is predictable and bounded before the system begins its execution in order to avoid unpredictable behaviors.

RV has receiving increasing attention in the real-time community in the past decade, with clear focus on relaxing the burden of the verification intensive tasks using deductive verification and model-checking. Deductive verification tends to get undecidable results when reasoning about time (the "undecidable satisfiability problem" for certain logic fragments), and systems tend to scale poorly when the model size grows (the "state space explosion problem").

Design of a decidable verification method to reason with explicit time properties (i.e., duration properties) at runtime is the main problem. It should be capable to describe polynomial inequalities mixed with temporal order of propositions using a formal logic in order to deal with hard real-time systems at the design phase. Moreover, it requires a separation of which properties classic model-checking is unfeasible to treat, due to the need of total coverage of the model, and what properties could be addressed statically using deductive approaches. RV only deals with one execution trace, hence it amounts to the "word acceptance" problem rather than the "emptiness check" problem as in model-checking.

Embedded real-time systems could be rather complex if control routines are considered and different numeric methods such as *proportional–integral–derivative* (PID) controllers [Åström and Hägglund, 2006], *extended Kalman filter* (EKF) [Julier and Uhlmann, 2004] are involved. For the majority of these cases, we cannot assume that fully describing the behaviors with polynomial inequalities is enough. A potential solution is to deal with well behaved fragments and if possible put on top of it other theories. This means that output of tools to discretize control models can be verified at the level of the discretization instead of at the design phase, feature that is addressed by dynamic logic [Platzer, 2008, Harel et al., 2000], temporal interval logic [Chaochen et al., 1993] and/or hybrid logic [Platzer, 2007, Blackburn and Tzakova, 1999, Blackburn and Seligman, 1995].

1.2 Summary of Research Contribution

Considering the potential solution identified in the end of the previous section, we believe that the polynomial description can be enough for the majority of the cases, rendering them verifiable. More precisely, we set out to provide evidence for the following statement:

Thesis. *Runtime verification of duration properties for hard real-time systems can be made through the use of the synthesization of a fragment of* Metric temporal logic with durations *(MTL-$\int$).*

We will support this statement by a set of techniques and tools for synthesization of monitors from a fragment of MTL-$\int$ and by a correct-by-construction implementation of the monitor integration on the target system. We developed a three-valued semantics for a fragment of MTL-$\int$ to deal with incomplete trace evaluation [De Matos Pedro et al., 2017, 2015a]. For that formal language, we introduce two synthesis algorithms: one for monitoring synthesis based on the theory of lists; and other for synthesis of satisfability problems for *satisfiability modulo theories* (SMT) solvers based on non interpreted functions with equality, arrays and non-linear arithmetic, including the use of quantifier elimination tactic. In case of monitoring synthesis, we proceed before synthesis by applying a simplifcation algorithm in order to remove and partially solve the quantifiers from formulas in the proposed fragment of MTL-$\int$ [De Matos Pedro et al., 2014b]. After that, the new monitoring algorithm will be ready to be executed. We also provided a mechanism to generate the monitor architecture according to the desired settings in order to be embedded in the target system [De Matos Pedro et al., 2014a]. The synthesis algorithm for SMT solvers is also presented as a first step to solve fundamental problems of hard real-time systems [De Matos Pedro et al., 2016, 2015b]. In this thesis we also provide the validation of the proposed techniques using an empirical use case about the schedulability analysis of hard real-time systems, and a set of use cases for the autopilot stack Px4 [Meier et al., 2015].

In addition, we have implemented a tool and a library that have come out of our research efforts and both are now available to the public. They are rmtld3synth [De Matos Pedro, 2018], a tool for synthesization of monitors and their respective safe inclusion, and RTMLib [De Matos Pedro, 2016] the library to aid the monitor execution.

1.3 Overview of Thesis

This thesis is organized into six chapters, corresponding the three core sections to the RV technique, the RV framework, and the practical evaluation of the technique. To accommodate readers, we provide a comprehensive introduction in Chapter 2 of the terminologies, notations, and techniques that are used extensively throughout the remainder of the thesis. The context for our research contribution with a discussion of related work in hard real-time embedded systems, languages and RV is also presented.

Chapter 3 describes a new mechanism for RV of hard real-time systems regarding duration

properties, based on a decidable fragment of MTL-$\int$ and a three-valued abstraction of this fragment. The fragment allows for expressing quantified formulae, and is adequate for quantifier elimination: we give an algorithm for the simplification of formulas containing quantifiers and free logic variables. Intuitively, we abstract our fragment into *first order logic of real numbers* ($\text{FOL}_{\mathbb{R}}$) to obtain quantifier-free formulas.

Chapter 4 provides a compositional framework that allows us to make assumptions about the time isolation between components as well as the response times of the monitors. We apply this notion to components with different criticality assurances, and whose specific requirements shall be guaranteed statically and dynamically through schedulability analysis and runtime monitoring, respectively. To guarantee these frameworks' assumptions we use the proposed fragment to analyze the schedulability of the *compositional monitoring framework* (CMF), and to statically check the maximum response times of each of the generated monitors.

Chapter 5 describes the practical evaluation of the proposed technique at level of both static and dynamic verification. By static we mean a formalization of a set of rules of a system resource usage as well as the claim of the resolution of the schedulability decision problem for periodic resource models using a formal language. As dynamic we consider the uncertainty monitoring and the practical case study of an autopilot. Considering that the adopted formalism supports an explicit notion of time by means of inequalities, durations and quantification over these formulas, it increases the expressiveness of classic temporal logic to deal with explicit timing settings as we point out here using practical evaluation experiments. Given the evaluation procedure that draws verdicts, we show the importance of such existence in the context of hard real-time systems by ensuring that a monitor always terminates and gives a result.

Finally, Chapter 6 discusses direction for future work in RV, including different synthesis mechanisms targeting embedded systems which have so restricted resources as well as different simplification techniques that may be adopted to use before submitting the problem for SMT solvers.

Chapter 2

Background and Related Work

The identification and formal description of the inherent behavior of hard real-time systems are two fundamental steps for establishing the verification process of these systems. Concerning identification, we characterize those systems and classify their schedulability problems. Regarding the formal description, we justify the necessity, and present the languages, to formally describe them.

Although specification languages and models for those systems are scarce, they are crucial to address the design of new verification approaches, particularly when it comes to *Runtime Verification* (RV). RV may be able to draw verdicts from more expressive formalisms than static formal verification may currently perform, even though RV deals exclusively with past executions and ideally reduces the burden for the software designer.

In this chapter, we give an overview of the properties of hard real-time systems, the formal description of available languages for these systems, and we then describe the collection of the *state of the art* in RV as well as the related work.

2.1 Real-Time Systems

RTS are those systems that are subject to timing constraints as well as resource constraints. Consequently, the correctness of such systems depends on both time and functional aspects where resource constraints may be included. According to [Burns and Wellings, 2009], real-time systems can be distinguished from other systems, in general, when failure to respond (or to react to a stimuli) can be considered non problematic. In [Mall, 2009], the author describes the *response time* as a distinctive feature of real-time systems – although for other authors this may be important, it is not crucial.

Real-time systems are typically divided into *soft real-time systems* and *hard real-time systems*. In soft real-time systems, missing a deadline degrades the performance (in average). For instance, dropping video frames while streaming a video conference may be inconvenient for the remote viewers, but no permanent harm is done. In hard real-time systems, deadlines cannot be missed. For instance, an orbital satellite controller is a hard real-time system since missing a deadline may cause the satellite to fail its orbit (wherever it occurs). In such systems, deadlines must be kept even under worst-case scenarios.

There are many interpretations of the exact nature of a real-time system since each author proposes a new one [Davis and Burns, 2011, Sha et al., 2004]. Nevertheless an important one is ≪in real-time computing the correctness of the systems depends not only on the logical result of the computation but also on the time at which the results are produced.≫ ([Stankovic, 1988]).

Real-time systems span a considerable range of application domains such as process control systems (e.g., a bottle filling assembly line), manufacturing systems (e.g., a production control system), embedded systems (e.g., an onboard satellite computer), and multimedia systems in general (e.g., a video streaming system), among many others. A few key characteristics distinguish them from the more general-purpose systems.

- *Time constraints*: crucial to ensure deadlines, execution times (or durations), and delays. For instance, deadlines restrict the time instant at which a process needs to be concluded;

- *Correctness criterion*: this notion applies to both non real-time systems and real-time systems. For real-time systems, this criterion differs from the one used in the context of traditional systems, since correctness here implies functional and temporal correctness. A functionally correct result produced after the deadline is considered as incorrect;

- *Support for numerical computation*: the notion required for hybrid systems support (e.g., control activities; a power plant management system). Real-time systems are often dynamic systems where at discrete points in time some timing constraints are required, but their behavior is a mix between discrete and dynamic systems;

- *Safety-Criticality*: denotes a mix between safety and reliability of systems. In traditional systems, *safety* and *reliability* are not combined. A system is considered safe when it does not cause any damage or injury even when it fails; reliability on the other hand, states that a system can operate for a long time without exhibiting any failure; and

- *Large and complex*: refers to the size and complexity of a system. While a small program may not have significant problems since it is simple in its essence, the same does not occur when developing a larger one.

Other characteristics are also applicable. However, they are not directly related to real-time systems but may be considered as *extension* features. For instance, time constraints characterize real-time systems directly, but a real-time system may, or may not, be distributed. Such new features or extensions are described as follows:

- *Reactive*: describes the capacity of the system to react to external stimuli, producing a feedback to the environment whenever the system evolves;
- *Concurrent*: consists of many parallel/concurrent interaction activities that should be handled at the same time, i.e., several coexisting external elements with which the computer program must interact simultaneously;
- *Distributed*: a notion of different components of the system being naturally distributed across spread physical locations;
- *Embedded*: represents the notion of custom-made independent systems which implement specific control functions. Usually, these are known as real-time embedded systems.[1];
- *Component criticality*: represents the cost of a component failure. Real-time systems may have components (or processes) of different criticalities. This introduces an analysis of how critical are the results produced by each component related to the proper functioning of the system;
- *Stability*: states that a system, even under overload conditions, complies to the timing constraints for the high criticality components; and
- *Fault-Tolerant*: characterizes the ability to avoid a system entering a faulty state. Under catastrophic scenarios, the system shall detect those states and continue operating normally (or even in degraded mode) rather than shutting off abruptly.

Note that any real-time system can exhibit one or more of these features, as they provide a coherent and congruent mix of characteristics.

One of the central issues in real-time systems is the mechanism to handle multiple interacting activities (e.g., tasks), guaranteeing their timing constraints. This is called *real-time*

[1]Note that embedded systems are becoming more and more complex and generic (e.g. a mobile phone; IoT home devices), therefore this distinction is starting to be fuzzy.

scheduling and it is a very active area of research. Tasks can be seen as abstract types which are used to denote components of code to be executed over certain constraints. They are triggered when an event occurs (e.g., pressing a power off button, or even when a kitchen robot see some stairs and avoid a faulty situation). These timing constraints can be seen as a time restriction of code execution. In the following paragraphs, tasks and timing constraints are classified.

Tasks. Real-time tasks can be classified as *hard*, *firm* or *soft*. These terms characterize their dependence on and consequences of a deadline miss. It is not necessary that all tasks of a real-time system belong to the same class. A hard real-time task is one that is constrained to produce its results within certain predefined time bounds. A firm real-time task, unlike hard real-time tasks, does do not fail when a timing constraint is not satisfied (e.g., video conferencing), but there is no value in delivering the result after the deadline. In a soft real-time task, timing constraints can be expressed in terms of the average response time, and results have some value, although limited, after the deadline.

Moreover, real-time tasks can be characterized as being *periodic*, *sporadic* or *aperiodic*. A periodic task is one that repeats within a fixed inter-arrival time; a sporadic task is one that recurs at random instants (it has a dynamic inter-arrival time with a minimum interval); and an aperiodic task is one that is similar to a sporadic task but has no minimum or maximum inter-arrival time.

Timing Constraints. Timing constraints may be described by events (e.g., the occurrence of an input in a system such as an engine start action). These events characterize the state changes of a system. Such systems can also be named as Discrete Event Systems (DES) [Cassandras and Lafortune, 2008]. The events generated by real-time systems can be classified as *stimulus events* or *response events*: the stimulus events are generated by the environment where a system run and acts on it; the response events are usually produced by the system in response to some stimulus of the environment (i.e., stimulus events). The timing constraints can be formulated through these type of events and classified by three constraints: *delay*, *deadline*, and *duration*. As the name suggests a delay d is the measure given by the time difference of two events e_1, e_2 greater or equal to the value d, $t(e_2) - t(e_1) \geq d$; a deadline is the bound of time b between two events such as $t(e_2) - t(e_1) \leq b$; and a duration dr corresponds to the inter-arrival time between two consecutive events, $t(e_2) - t(e_1) = dr$. Timing constraints are in their essence timing behaviors of real-time systems.

A task is instantiated multiple times and each instantiation is commonly denoted as a

job. The deadlines of real-time jobs can be *relative* to one time instant (e.g., the arrival of a stimulus event) or *absolute* (from when the system started executing). The response time is defined by the time duration between the job release and the instant that the task finalizes its execution.

Scheduling algorithms normally target uniprocessor, multiprocessor, and distributed systems. Several major abstractions can be applied between uni/multi-processor systems and distributed systems. They have in their essence major delays and spatial positions.

Although the main focus of this thesis is not on real-time scheduling, we have to provide the classic schedulability analysis of *periodic resource models* in order to introduce the meaning of resources in the context of the next chapters.

2.1.1 Periodic Resource Models

Let us assume a *tasks set* $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, such that $n \in \mathbb{N}^+$ is the identifier of periodic tasks, and $\tau_i = (p_i, e_i)$ with p_i and e_i being, respectively, the period and the worst-case execution time of the periodic task τ_i; and a set of *periodic resource models* $\Omega = \{\omega_1, \omega_2, ..., \omega_m\}$ with

$$\omega_j = (\mathrm{T}, \pi, \theta, rm),$$

where $\mathrm{T} \subseteq \Gamma$, π is the *replenishment period*, θ is the *server budget*, and rm is the *rate monotonic* scheduling policy.

The schedulability analysis for periodic resource models was first provided by Shin and Lee [Shin and Lee, 2003, 2008]. The authors formulate an analysis based on *resource model supply*. The *supply bound function* $sbf_\omega(t)$ is defined to calculate the minimum resource supply for every interval of length t as follows:

$$sbf_\omega(t) = \begin{cases} t - (k+1)(\pi - \theta) & \text{if } t \in \mathcal{I}, \\ (k-1)\theta & \text{otherwise,} \end{cases}$$

where $\mathcal{I} = [(k+1)\pi - 2\theta, (k+1)\pi - \theta]$. The value k is given by

$$k = \begin{cases} x & \text{if } x > 1 \\ 1 & \text{otherwise} \end{cases},$$

where $x = \left\lceil \frac{t-(\pi-\theta)}{\pi} \right\rceil$.

For an arbitrary set of tasks τ and a rate monotonic scheduling policy, Lehoczky et al. [Lehoczky et al., 1989] proposed a demand-bound function $dbf_{rm}(\tau, t, i)$ that computes

the worst-case cumulative response demand of a task $\tau_i \in \tau$ for any interval of length t. It is defined by

$$dbf_{rm}(\tau, t, i) = e_i + \sum_{\tau_k \in \gamma_\tau(i)} \left\lceil \frac{t}{p_k} \right\rceil . e_k,$$

where $\gamma_\tau(i) = \{\tau_1, ..., \tau_i\}$ is a function that returns a set of tasks with higher-priority than (and including) task τ_i, and τ is a periodic task set. The *demand-bound function* for resource models is the same since the set of tasks is schedulable using the rate monotonic policy. This means that the supply of a resource model must be greater than the demand of the set of tasks that a resource model contains.

The tasks set T of a resource model is said to be schedulable according to a rate monotonic policy if, and only if,

$$\forall \tau_i \in \mathrm{T},\ \exists t_i \in [0, p_i] \text{ s.t. } dbf_{rm}(\mathrm{T}, t_i, i) \leq sbf_\omega(t_i).$$

2.2 Languages and Logics

Although any property of a system may be expressed in natural language, it is hard to ensure that someone else will understand exactly what it means. Natural languages are very expressive but, at the same time, imprecise. On the other hand, formal languages are not very expressive but they are very precise, and do not allow for multiple interpretations of the same concept.

Temporal logic is known as a language that is adequate for expressing temporal properties such as *liveness* and *safety*. Safety properties ensure that a program does not do something bad. Liveness properties ensure that the program does eventually something good.[2] Temporal logics have been used as a formalism for specifying qualitative ordering constraints on the observable traces. The best-known logic is *linear temporal logic* (LTL) [Pnueli, 1977]. A formula in this logic is built from atomic propositions, standard boolean operators, and modal operators. Nevertheless, LTL is not adequate for real-time systems specification. A run of a real-time system needs to be modeled with *timed interval sequences* or as *flows* with domain in $\mathbb{R}_{\geq 0}$.

The most widely known extension of LTL for dealing with real-time is *metric temporal logic* (MTL) in which the modalities of LTL are augmented with timing constraints [Alur and Henzinger, 1992b]. A common modality is called *until* and is denoted by U. Usually, temporal operators can be strict (when they do not constrain the current instant) or not, and matching (when they require their two arguments to hold together) or not. Intuitively,

[2] There are other properties, but they are out of the scope of this thesis.

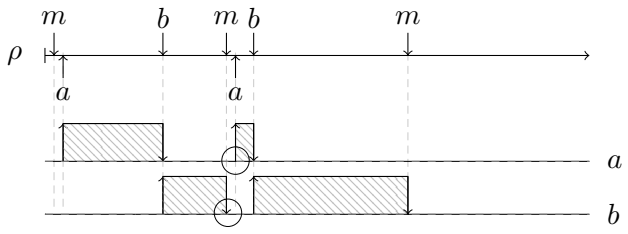


Figure 2.1: Evaluation of propositions m, a, b over the trace ρ.

$\varphi_1 \,\mathrm{U}_{<t}\, \varphi_2$ is interpreted by true if along the execution trace (from 0 to t, excluding t), there exists a point where φ_2 holds, and such that all intermediate points satisfy φ_1. In case φ_2 is true then the formula is evaluated to true. Intuitively, we are describing the point-wise semantics of the until operator, that is strict and non-matching.

Common shorthands for metric operators are *always* ($\Box$ or A) and *eventually* ($\Diamond$ or E).

Example 1. *Let us assume that the symbol m is periodically released at each 20 time units, the trace ρ begins at $t = 0$, and that the until operator is strict and non-matching. In Figure 2.1, we can observe that $a \,\mathrm{U}_{<16}\, b$ and $a \,\mathrm{U}_{<20}\, b$ are evaluated to false. However, if we specify the formula*

$$\Box_{<u}\ m \rightarrow \Diamond_{<2}\ a \,\mathrm{U}_{<16}\, b,$$

for $u = 40$, the evaluation is true. Intuitively, we are describing that for each occurrence of the event m in the interval $[0, 40[$, in at least 2 time units the event a occurs, and that the event a holds until the event b holds in at least 16 time units. Note that if we replace u by 41, the formula is evaluated to false. The third occurrence of the m symbol does not hold, neither do the symbols a and b occur further ahead.

MTL formulas can be interpreted over a variety of temporal models such as discrete (e.g., $\mathbb{N}$, $\mathbb{Z}$) [Emerson, 1990, Alur and Henzinger, 1993] and dense (e.g., $\mathbb{R}$) [Hirshfeld and Rabinovich, 2004, Bouyer et al., 2010, Souza and Prabhakar, 2007, Furia and Rossi, 2007] time domains. Metric operators defined over discrete time can be regarded as simple *syntactic sugar*, since they are a succinct way of expressing metric constraints that can be encoded using the LTL's *next* modality. Dense-time MTL operators are commonly classified in terms of *pointwise* and *continuous* semantics. The pointwise semantics is evaluated along possibly infinite sequences of timed words, i.e., sequences of pairs

$$(e_0, t_0)(e_1, t_1)\ldots,$$

where the e_i are events/propositions belonging to an alphabet Σ and $t_i \in \mathbb{R}_{\geq 0}$ are the occurrence time instants of the events e_i. The continuous semantics is evaluated over possibly infinite signals. Given a set of propositions P, a signal is a function $f : \mathbb{R}_{\geq 0} \rightarrow 2^P$ mapping $t \in \mathbb{R}_{\geq 0}$ to the set $f(t)$ of propositions holding at time t. A restriction of the

continuous semantics for evaluating timed interval sequences is also known as an *interval-based semantics*, or in other words, a *continuous semantics* with finite variability. Timed interval sequences are sequences of pairs

$$(e_0, l_0)(e_1, l_1)\ldots,$$

where the l_i are contiguous, non-overlapping intervals with real or rational bounds, forming a sequence of intervals of $\mathbb{R}_{\geq 0}$.

The majority of real-time systems operate in a dense time domain and states are always changing at any time instant. Even if it may be possible to get infinitely many changes over a fixed interval of time as the case of control systems, this will give us undecidable results. As explained by Henzinger and colleagues [Henzinger et al., 1992] many verification methods are based on the assumption that states are only observed at integer points (also called *digitization*). Here, we are talking about digital systems, where such infinitely many changes cannot occur. *Metric temporal logic with durations* (MTL-$\int$) is thus appropriate for reasoning about such systems. However, the verification of digital systems does not require the expressive power of continuous ($\mathbb{R}$) semantics. Instead, it may be sufficient to restrict the input model to *timed interval sequences*.

MTL-$\int$ extend expressiveness of MTL with fragments of classic logic, including *first order logic of real numbers* ($\text{FOL}_{\mathbb{R}}$). Nevertheless, we do not have a hybridization [Blackburn and Tzakova, 1999], since we have terms and formulas separated, and quantification only occurs over relation $<$ (a predicate in $\text{FOL}_{\mathbb{R}}$) containing terms as argument. MTL-$\int$ is more expressive than $\text{FOL}_{\mathbb{R}}$. Moreover, lambda calculus can encode fragments of temporal logic without making use of a proper lambda calculus temporal extension as proposed in [Davies, 2017]. Lambda expressions will be described after introducing MTL-$\int$ and $\text{FOL}_{\mathbb{R}}$.

2.2.1 *Metric temporal logic with durations* (MTL-$\int$)

MTL-$\int$ is more expressive than *duration calculus* (DC) [Lakhnech and Hooman, 1995, Chaochen et al., 1993], but is undecidable since the relation over terms or the term function may itself be undecidable. DC is based on interval logic and includes the chop modality instead of the until modality as in temporal logic. This constructing operator allows us to find a point in time where an interval can be split into two sub-intervals. Implicitly, this express a temporal bound over liveness properties. Although DC is able to deal with liveness properties as in MTL, the inverse chop modality shall be considered. Let us begin by briefly reviewing MTL-$\int$.

Definition 1. Let $\mathcal{P}$ be a set of propositions and $\mathcal{V}$ a set of logic variables. The syntax

of MTL-$\int$ terms η and formulas φ is defined inductively by

$$\eta ::= \alpha \mid x \mid f(\eta_1, \ldots, \eta_n) \mid \int^{\eta} \varphi$$
$$\varphi ::= \mathit{true} \mid p \mid R(\eta_1, \ldots, \eta_n) \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \varphi_1 \,\mathrm{U}_{\sim\gamma}\, \varphi_2 \mid \varphi_1 \,\mathrm{S}_{\sim\gamma}\, \varphi_2 \mid \exists x\, \varphi$$

where $\alpha \in \mathbb{R}$, $x \in \mathcal{V}$ is a logic variable, f a function symbol of arity n, $\int^{\eta} \varphi$ is the duration of the formula φ in an interval, $p \in \mathcal{P}$ is an atomic proposition, U and S are temporal operators with $\sim \in \{<, =\}$, $\gamma \in \mathbb{R}_{\geq 0}$, and $R(\eta_1, \ldots, \eta_n), \varphi_1 \vee \varphi_2, \neg\varphi$, and $\exists x\, \varphi$ are defined as usual.

Furthermore, we will use the following abbreviations: $\varphi \wedge \psi$ for $\neg(\neg\varphi \vee \neg\psi)$, $\varphi \to \psi$ for $\neg\varphi \vee \psi$, $\Diamond_{\sim\gamma}\, \varphi$ for $\mathit{true}\,\mathrm{U}_{\sim\gamma}\, \varphi$, and $\Box_{\sim\gamma}\, \varphi$ for $\neg(\mathit{true}\,\mathrm{U}_{\sim\gamma}\, \neg\varphi)$.

An *observation function* σ of length $\delta \in (\mathbb{R}_{\geq 0} \cup \{\infty\})$ over $\mathcal{P}$ is a function from $\mathcal{P}$ into the set of functions from the interval $[0, \delta)$ into $\{\mathsf{tt}, \mathsf{ff}\}$. The length of σ is denoted by $\#\sigma$. A *logical environment* is any function $\upsilon : \mathcal{V} \to \mathbb{R}_{\geq 0}$. For any υ, $x \in \mathcal{V}$ and $r \in \mathbb{R}$, we will denote by $\upsilon[x \mapsto r]$ the logical environment that maps x to r and every other variable y to $\upsilon(y)$. The following auxiliary definition will be used in the interpretation of the duration of a formula.

Definition 2 (MTL-$\int$ semantics)**.** The truth value of a formula φ will be defined relative to a model (σ, υ, t) consisting of an observation σ, a logical environment υ, and a time instant $t \in \mathbb{R}_{\geq 0}$. We will write $(\sigma, \upsilon, t) \models \varphi$ when φ is interpreted as true in the model (σ, υ, t). Terms and formulas will be interpreted in a mutually recursive way. First of all, for each formula φ, observation σ and logical environment υ, the auxiliary indicator function $1_{\varphi(\sigma,\upsilon)} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is defined as follows, making use of the satisfaction relation:

$$1_{\varphi(\sigma,\upsilon)}(t) = \begin{cases} 1 & \text{if } (\sigma, \upsilon, t) \models \varphi, \\ 0 & \text{otherwise.} \end{cases}$$

The value $\mathscr{T}[\![\eta]\!](\kappa,\upsilon)\, t$ of a term η relative to a model can then be defined. A *Riemann integral* [Gordon, 1994] of $1_{\varphi(\sigma,\upsilon)}$ is used for the case of a duration $\int^{\eta} \varphi$:

$$\begin{aligned}
\mathscr{T}[\![\alpha]\!](\sigma, \upsilon)\, t &= \alpha \\
\mathscr{T}[\![x]\!](\sigma, \upsilon)\, t &= \upsilon(x) \\
\mathscr{T}[\![f(\eta_1, \ldots, \eta_n)]\!](\sigma, \upsilon)\, t &= f\left(\mathscr{T}[\![\eta_1]\!](\sigma, \upsilon)\, t, \ldots, \mathscr{T}[\![\eta_n]\!](\sigma, \upsilon)\, t\right) \\
\mathscr{T}\left[\!\!\left[\int^{\eta} \varphi\right]\!\!\right](\sigma, \upsilon)\, t &= \begin{cases} \int_t^{t+\mathscr{T}[\![\eta]\!](\sigma,\upsilon)\, t} 1_{\varphi(\sigma,\upsilon)}(t')\, dt' & \text{if } (*) \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

where $(*)$ means that $1_{\varphi(\sigma,\upsilon)}$ satisfies the Dirichlet condition [Lakhnech and Hooman, 1995, p.7][3] and the sub-term $\mathscr{T}[\![\eta]\!](\sigma,\upsilon)\,t$ is non-negative, otherwise the function is non Riemann integrable. The satisfaction relation in turn is defined as:

$$
\begin{array}{lll}
(\sigma,\upsilon,t) \models p & \text{iff} & \sigma(p)(t) = \mathtt{tt} \text{ and } t < \#\sigma \\
(\sigma,\upsilon,t) \models R(\eta_1,\ldots,\eta_n) & \text{iff} & R(\mathscr{T}[\![\eta_1]\!](\sigma,\upsilon)\,t,\ldots,\mathscr{T}[\![\eta_n]\!](\sigma,\upsilon)\,t) \\
(\sigma,\upsilon,t) \models \varphi_1 \vee \varphi_2 & \text{iff} & (\sigma,\upsilon,t) \models \varphi_1 \text{ or } (\sigma,\upsilon,t) \models \varphi_2 \\
(\sigma,\upsilon,t) \models \neg\varphi & \text{iff} & (\sigma,\upsilon,t) \not\models \varphi \\
(\sigma,\upsilon,t) \models \varphi_1\,\mathrm{U}_{\sim\gamma}\,\varphi_2 & \text{iff} & \text{there exists } t' \text{ such that } t \le t' \sim t+\gamma,\ (\sigma,\upsilon,t') \models \varphi_2, \\
 & & \text{and for all } t'',\ t < t'' < t',\ (\sigma,\upsilon,t'') \models \varphi_1 \\
(\sigma,\upsilon,t) \models \varphi_1\,\mathrm{S}_{\sim\gamma}\,\varphi_2 & \text{iff} & \text{there exists } t' \text{ such that } t-\gamma \sim t' \le t,\ (\sigma,\upsilon,t') \models \varphi_2, \\
 & & \text{and for all } t'',\ t' < t'' < t,\ (\sigma,\upsilon,t'') \models \varphi_1 \\
(\sigma,\upsilon,t) \models \exists x\,\varphi & \text{iff} & \text{there exists an } r \in \mathbb{R} \text{ such that } (\sigma,\upsilon[x \mapsto r],t) \models \varphi
\end{array}
$$

Note that the semantics of the until operator is strict and non-matching [Bouyer et al., 2010].

Figure 2.2a intuitively illustrates the use of the MTL-$\int$ language. From Figure 2.2b we can conclude that the formula $\forall x\ \int^x(\epsilon_\beta) \le \int^x \epsilon_\beta \vee \epsilon_\alpha < x$ in the finite interval $[0, 64)$ is interpreted as true. Note that $\forall x\ \phi$ is a shorthand for $\neg\exists\neg\phi$.

2.2.2 *first order logic of real numbers* ($\mathbf{FOL_{\mathbb{R}}}$)

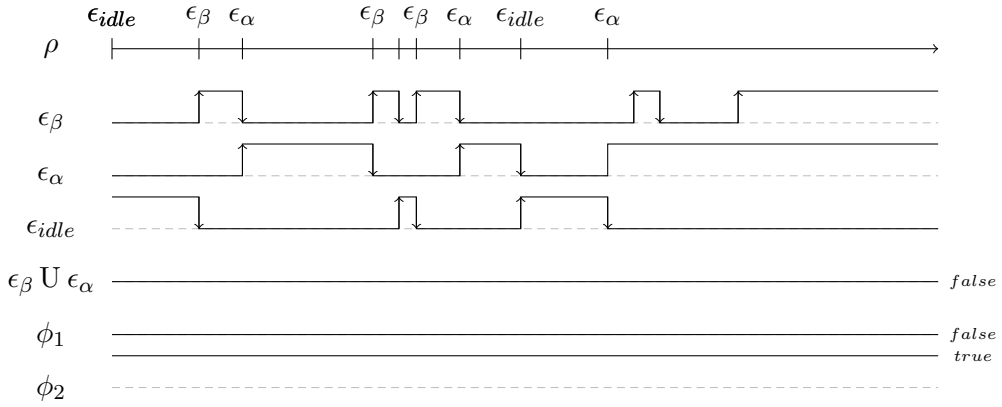
$\mathrm{FOL}_{\mathbb{R}}$ commonly denotes the first order logic defined over the structure $(\mathbb{R}, <, +, \times, 1, 0)$ [Jovanović and de Moura, 2013]. $\mathrm{FOL}_{\mathbb{R}}$ formulas, also known as Tarski formulas [Tarski, 1995], are boolean combinations of polynomial equalities and inequalities. We define $\mathbb{Z}[x]$ by $\bigcup P_n$ as a ring of polynomials with one variable x, where $P_0 = \mathbb{Z}$, and $P_n = xP_{n-1} + P_{n-1}$.

Definition 3 ($\mathrm{FOL}_{\mathbb{R}}$)**.** A polynomial $f \in \mathbb{Z}[\mathbf{y}, x]$ is of the form

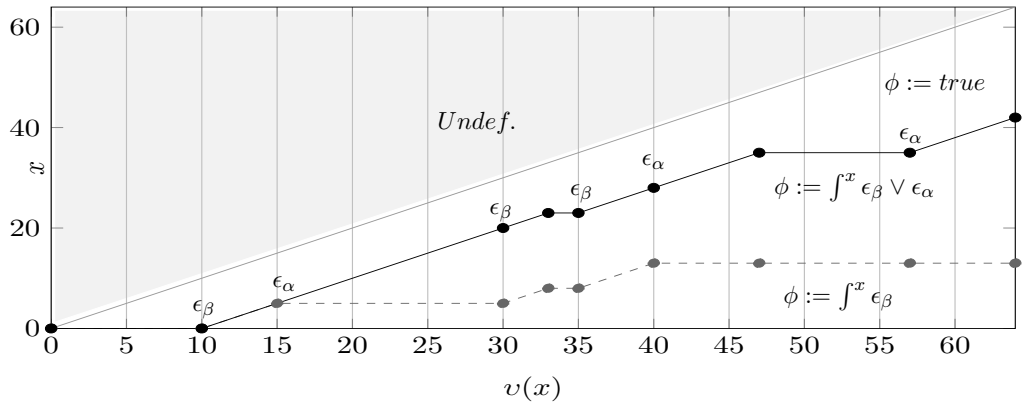
$$f(\mathbf{y}, x) = a_m \cdot x^{d_m} + a_{m-1} \cdot x^{d_{m-1}} + \cdots + a_1 \cdot x^{d_1} + a_0,$$

where $0 < d_1 < \cdots < d_m$, and the coefficients a_i are in $\mathbb{Z}[\mathbf{y}]$ with $a_m \neq 0$. A polynomial constraint F is of the form $f \triangledown g$ where f, g are polynomials and $\triangledown \in \{<, \le, =, \neq, \ge, >\}$. We denote the polynomial constraint that represents the negation of a constraint F by $\neg F$. A *clause* of polynomial constraints is a disjunction $F_1 \vee \cdots \vee F_n$ of $n \in \mathbb{N}$ polynomial constraints. Note that in this definition we do not consider *roots* of polynomials.

[3] A function is said to satisfy the Dirichlet condition if and only if for any bounded interval I, it is bounded in I and has a finite number of discontinuity points in I.



(a) A diagram containing: a path ρ; three event releases ϵ_β, ϵ_α, and ϵ_{idle}; and the respective truth value of the logic formulas $\epsilon_\beta \,\mathrm{U}\, \epsilon_\alpha$, $\phi_1 := \int^{30} \epsilon_\beta \vee \epsilon_\alpha \leq 10$, and $\phi_2 := \int^{30} \epsilon_\beta \leq 10$.



(b) The graph depicts the formula $\int^x \epsilon_\beta$ and $\int^x \epsilon_\beta \vee \epsilon_\alpha$ which allows us to visually check the formula $\forall x \;\; \int^x \epsilon_\beta \leq \int^x \epsilon_\beta \vee \epsilon_\alpha < x$ in the finite interval $[0, 64)$.

Figure 2.2: Diagram of a path (a) and respective duration computation (b)

Example 2. *Let us now consider the polynomial inequality* $50 - x^2 \cdot y < 10$. *It can be expressed using the pattern of the Definition 3 by*

$$50 < (1 \cdot y + 0) \cdot x^2 + 0 \cdot x^1 + 10,$$

where coefficient a_2 *is replaced by the monomial* y. *Considering* a_2 *equals to* $1 \cdot y^2 + 1 \cdot y^1 + 0$, *we get* $50 < a_2 \cdot x^2 + 0 \cdot x^1 + 10$ *that is equivalent to* $50 < x^2 \cdot (y^2 + y) + 10$.

2.2.3 Lambda expressions (λ-expressions)

The lambda calculus, commonly denoted by λ-calculus, was introduced in the 1930s by Alonzo Church [Church, 1941]. It consists of a notation for describing mathematical functions and programs, and a functional abstraction that captures some of the essential

common features of a wide variety of programming languages [League, 2000]. It is commonly described as the smallest universal programming language, since it is equivalent to Turing machines. However, λ-calculus is focused on the transformation rules and single function definition scheme, instead of the shape of the actual machine implementing them. As such it is an approach more related to software than to hardware.

A λ-term is either a variable $x \in Var$, where Var is a countably infinite set of variables; an application of a function e_0 applied to an argument e_1, usually written $e_0\ e_1$; or a lambda abstraction, $\lambda x.e$ representing a function with input parameter x and body e. Formally, lambda expressions are inductively defined by

$$e ::= x \mid \lambda\, x.e \mid e_0\, e_1$$

where the metavariable e represents a λ-calculus term.

An expression can be surrounded with parenthesis for clarity, and we use the notation with "."s to avoid the proliferation of multiple lambdas, each one with one argument. For instance, $\lambda x_1, \ldots x_n.M$ is equivalent to $(\lambda x_1(\ldots(\lambda x_n M)\ldots))$, where M is the body of the abstraction. We assume that lambda abstractions associate to the right, and applications to the left, i.e., $MN_1 \ldots N_n$ is equivalent to $(\ldots(MN_1)\ldots N_n)$. Note that λ acts as a variable binder in a similar way to the quantifiers $\exists$ and $\forall$ in *predicate calculus* and $\int \ldots dx$ in *integral calculus*.

We begin by describing the meaning of the β reduction ($\longrightarrow_\beta$)

$$(\lambda x.M)N \longrightarrow_\beta M[N/x],$$

where $M[N/x]$ can be read "replace free occurrences of x in M by N". The α-rule is defined by

$$\lambda x.M = \lambda y.M[y/x] \text{ and } y \text{ is not a free variable of } M.$$

This rule captures the fact that a bound variable can be replaced by any other free variable. The reduction denoted by $\longrightarrow_\beta^*$ is the transitive and reflexive clousure of $\longrightarrow_\beta$.

Substitution suffers from the problem of "variable capture". It can be solved using different approaches. A simple one is to replace the bounded variables in certain circumstances as in [League, 2000, Hindley and Seldin, 2008]. For instance, to evaluate $\lambda y.(\lambda x.yx)(xz)$, we have that $(\lambda x.yx)[y/xz]$. Here, using the modern approach, we need to use the α-reduction to rename x and reduce $(\lambda w.yw)[y/xz]$ into $\lambda w.xzw$.

The concept of equality in λ-calculus is not the same as in most of mathematics where it is called *extensional* equality. Instead of including the assumption that for funtions f_1, f_2 with the same domain, for all x , $f_1(x) = f_2(x)$ implies that $f_1 = f_2$; we have that two

Combinator	λ-calculus term
I	$\lambda x.x$
K	$\lambda xy.x$
S	$\lambda xyz.xz(yz)$
B	$\lambda xyz.x(yz)$
C	$\lambda xyz.xzy$

Combinator	λ-calculus term
T	$\lambda xy.\mathbf{K}xy$
F	$\lambda xy.y$
N (NOT)	$\lambda p.p\mathbf{FT}$
O (OR)	$\lambda pq.ppq$
E (ITE)	$\lambda pab.pab$

Table 2.1: Standard and Boolean Combinators

terms are equal if they encode the same algorithm in some way. This does not means that if two programs compute the same mathematical function then they are the same program. Note that one of them may be more efficient than the other. The λ-calculus is then said to have *intensional* equality. Different extensions exist but they converge in the same results.

There are a diverse set of combinators. Combinators are lambda terms with no free variables. Informally, combinators are completely specified operations. Some of the special combinators are the substitute-and-apply operator **S**, the identity operator **I**, the constant operator **K**, the swap operator **C**, and the compose operator **B**. Church Booleans are other special combinators: the truth value true **T**, the truth value false **F**, the *if-then-else* (as know as ite) **E**, the *or* operator **O**, and the *not* operator **N**. All of them can be found in Table 2.1.

Example 3. *Let us now see an example using Boolean combinators and the if-then-else operator. Consider the term "if a then* **T** *else* **F***". Case when* $a =$ **T**, *we have* **E T T F** *equals to*

$$(\lambda pab.pab)(\lambda xy.x)(\lambda xy.x)(\lambda xy.y) = (\lambda xy.x)(\lambda xy.x)(\lambda xy.y) = (\lambda xy.x) = \mathbf{T}.$$

For $a =$ **F**, *we have* **E F T F** *equals to*

$$(\lambda pab.pab)(\lambda xy.y)(\lambda xy.x)(\lambda xy.y) = (\lambda xy.y)(\lambda xy.x)(\lambda xy.y) = (\lambda xy.y) = \mathbf{F}.$$

To sum up, λ-calculus, more properly the typed λ-calculus, is the basis of the well-known functional programming languages such as ML and OCAML [Rémy, 2002]. As such it may be an elegant theory to synthesize/encoding temporal logics for different purposes such as monitors and/or SMT solvers.

2.2.4 Related Work

At the beginning of the 1990s, real-time constraints have been added to temporal logics [Koymans, 1990, Alur et al., 1993], in order to extend this vocabulary with the specification

of quantitative timing constraints. A bewildering diversity of operators are used in timed temporal logics that introduce considerable variations on the decidability and expressiveness of properties. There are two well-established families of timed logics with linear time. The first one is characterized by modalities decorated with quantitative constraints and is named *timed propositional temporal logic* (TPTL). TPTL [Bouyer et al., 2010] makes use of quantification together with untimed temporal modalities and explicit constraints on time values. The second one that is characterized by the *freeze-quantification* is *metric temporal logic* (MTL). MTL uses the time interval constrained modalities "until" and "since".

Alur and Henzinger [Alur and Henzinger, 1994] investigated the expressiveness and decidability properties of timed logics MTL and TPTL. They showed that MTL can be easily translated into TPTL. Furthermore, they conjectured, giving an intuitive example, that TPTL is more expressive than MTL. In [Maler and Nickovic, 2004] a fragment of MTL for continuous signals is considered, which is intrinsically different from observing discrete signals in a continuous time domain.

Nevertheless, MTL and TPTL are both undecidable even for finite timed words. Thus, several restrictions have been proposed to obtain decidable sub-logics such as Bounded-MTL [Bouyer et al., 2008b] which has "bounded" intervals (its satisfiability EXPSPACE-complete), and *metric interval temporal logic* (MITL) [Alur et al., 1996] which is decidable in EXPSPACE. Subsets of TPTL are less studied; one of such logics, the constrained TPTL, can be found in [Pandya and Shah, 2010, Parys and Walukiewicz, 2009].

Logics suitable for expressing linear-time temporal properties of event timed sequences or timed resources are *timed linear-time temporal logic* (TLTL) [Bouyer, 2009] and *weighted metric temporal logic* (WMTL) [Bouyer et al., 2008a]. Moreover, the well-known branching-time temporal logic for timed words TCTL (UPPAAL's [Behrmann et al., 2006] underlying logic). Such logics are well suited for expressing simple time-bounded response properties in linear and branching time. For instance, several simple properties can be defined by these logics such as: an event a occurs in three time units, or even an event a consumes at least three energy units.

The temporal logic MITL is one of the most popular real-time extensions of LTL. The main modality of MITL is the timed until U_I where I is some non-punctual interval with integer or rational endpoints. The original version of MITL contained only future temporal operators, although past and future versions of MITL were proposed in [Alur and Henzinger, 1992b].

Nevertheless, none of these related logics deals with explicit time, i.e., when counting time is required. MTL-$\int$ and DC are the languages that better fit the requirement of embedded hard real-time systems. DC is an interval logic making use of a *chop* operator instead of

the common temporal modalities, and MTL-$\int$ is more expressive than DC. The excessive expressiveness of such languages makes them intractable. Neither DC or MTL-$\int$ is more convenient to describe embedded real-time systems. They are simply different languages within the same roots on temporal logic. However, we believe that intrinsic temporal modalities such as *until* and *since* inside the logic are more convenient and intuitive for dealing with RV.

2.3 Runtime Verification

The increasing pervasiveness of critical applications in the context of safety-critical systems leads us to state, according to [Baier and Katoen, 2008], the following sentence: "The reliability of safety-critical systems is a key issue in the system design process". The magnitude of real-time systems, as well as their complexity, grows apace, meaning that there are no longer small and standalone applications. Typically, such systems are embedded in a larger context where several other components and systems connect and interact. These systems become much more vulnerable to errors – the number of defects grows exponentially with the number of interacting system components. In particular, phenomena such as concurrency and non-determinism that are central to modeling real-time systems turn out to be very hard to handle with standard known techniques.

Formal verification have an inherent separation in two kinds of approaches: *deductive reasoning* [Makinson, 2012, Almeida et al., 2011], where techniques by logic deduction are applied (e.g., iterative theorem proving, automated theorem proving [Harrison, 2009]); and *model-based verification* where properties are checked for all execution traces (e.g., classical model checking [Clarke et al., 1999], probabilistic model checking [Baier and Katoen, 2008]). The latter will be the focus of this chapter since timed temporal logics, a known formalism for checking timed systems, are well suited for modeling real-time systems, and also because the RV concept is close to model checking techniques (i.e., a trace model instead of an automaton).

Real-time systems are systems where RV may play an important role, not only due to their high complexity, which makes several static approaches practically unfeasible in a foreseeable future [Zhu et al., 2009, Leucker and Schallhart, 2009, Falcone, 2010], but also due to their high dependence on temporal constraints (e.g., reachability becomes undecidable due to the time clock operations: addition, subtraction by a constant, etc.) [Norström et al., 1999, Fersman et al., 2007, Krcal et al., 2007, Burns and Wellings, 2009]. The research on techniques for these systems has been growing progressively along the recent years, due to a high need for reliable and safe development alternatives to static

approaches. Nonetheless, the trend towards new dynamic approaches has been higher for soft real-time systems rather than for hard real-time systems (by focusing essentially on the functional aspects).

The *Runtime Verification* (RV) technique monitors the behavior of a system to check its conformance to a set of desirable logical properties. Note that the RV literature mostly focuses on event-triggered solutions. Nonetheless, this monotonic event invocation introduces two major defects to the system under scrutiny, namely significant overhead, and unpredictability. These effects can however be eliminated by using more recent techniques such as event-based monitoring with predictive analysis [Zhu et al., 2009], and sample-based monitoring with predictive analysis as introduced by [Fischmeister and Ba, 2010, Bonakdarpour et al., 2011].

Runtime monitoring (or monitoring upon execution time) is based on the synthesis of monitors (dedicated blocks of source-code) in an automatic way from formal specifications. It can be deployed *offline* for debugging, or *online* for dynamically checking properties during execution. Offline monitoring is currently a slightly inactive research topic; it consists in collecting a program trace (i.e., an execution trace) which is afterwards analyzed to verify if the execution is in compliance with the specification or not. For the purposes of replay and analysis of the scheduling process offline monitoring may be used to capture from a system implementation some operations such as: system calls, interrupts, context switches, and state variables. Online monitoring, on the other hand, may for instance ensure, by checking upon execution, that when a plug-in is loaded dynamically by one application, its consumed resources shall not exceed the resources allowed by the host application. This can be performed via *inline* monitoring, where the monitoring is inserted into execution code as annotations (e.g., assertions), or else by *outline* monitoring, where the monitor executes as a separate concurrent process. In addition, outline monitors may be implemented by hardware, synthesized from high level formal specifications and executed on FPGAs, resulting in zero runtime overhead on the system's CPU [Goodloe and Pike, 2010]. Typically, RV involves a significant time penalty when a system is under execution, thereby some authors [Sankar and Mandal, 1993, Pellizzoni et al., 2008] propose that it is crucial to use multi-processor systems when a hardware monitoring approach is not used. Using a multi-processor allows the monitoring process to be performed concurrently on a different processor, without delays for the system under monitoring.

Predictive analysis of runtime monitors refers to the ability of ensuring that real-time concurrent systems under scrutiny are *sound*. Soundness means that the predictive analysis is able to detect, correctly, functional (or even concurrency) errors from observing execution traces.

In the last decades, several RV approaches have emerged, mainly for concurrent systems. These approaches are an alternative or a complement to the conventional methods (e.g., model checking [Clarke et al., 1999], theorem proving [Fitting, 1996], and testing [Hamlet, 2010]), and, as such, a lightweight manner to check the behavior of systems, even if only partially. Let us now give a formal definition of RV.

Definition 4. (Runtime Verification) RV is a verification technique that allows checking whether a *run* of a system under scrutiny satisfies or violates a given correctness property.

RV deals with the observation problem, it detects violations (or satisfactions) of specified properties that can (or cannot) be mitigated. A violation occurs when a system under scrutiny deviates from the required behavior of the system.

Runtime Monitoring. Runtime monitoring is a process that is able to enforce property checking for systems during execution time. By *system under monitoring* (SUM), we consider a *system under observation* (SUO) where its evolving execution is observed at selected points (along the execution time) and those observations are checked against the given specifications [Goodloe and Pike, 2010]. In a more general perspective, runtime monitoring can be viewed as a technique that allows to check *past finite execution trace* (PFET) of a system. As such, runtime monitoring may only observe *finite executions* (past observations), contrary to classical verification techniques (e.g., model checking) where the focus is only on *infinite executions*. Thus, an execution of a system may be viewed as a finite prefix of a possibly infinite execution, and is therefore considered a PFET. The notion of runtime monitor is established in a slightly more general form in Definition 5.

Definition 5. (Runtime Monitor) A runtime monitor is a process that reads a PFET and yields a certain verdict at execution time.

By verdict we mean, in abstract, a truth value from some truth domain. This domain can be commonly-valued *true* and *false*, three-valued *true*, *false* and *unknown*, or even yielding a probabilistic interval in $[0, 1]$.

The problem of RV, in its mathematical essence, can be reduced to answering the *word problem*, i.e., the problem of whether a given word is included in some language. Let $[\![\varphi]\!]$ denote the set of valid executions satisfying the property φ. The *word inclusion* problem consists in checking whether the execution w is an element of $[\![\varphi]\!]$. On the other hand, the *language inclusion* problem is more complex and undecidable in general (e.g., classical timed automata) [Alur and Dill, 1994, Alur et al., 1999].

Runtime monitoring has been applied to concurrent (or even soft real-time) systems in order to detect functional violations at runtime, and trigger system recovery actions when

a catastrophic error occurs. However, runtime monitoring can be applied to nonfunctional aspects of a system through constraints, such as: performance, time, costs/weights or even resources utilization. Currently, as far as we are aware, there are no monitoring frameworks for such constraints.

Let us now overview *logic*-based monitoring. In spite of the fact that runtime monitors typically only have finite execution traces available at some point in execution, this does not imply that logics for infinite traces such as LTL, *computation tree logic* (CTL), or even the *superset of CTL* (CTL*) cannot be adopted to (or restricted only to) analyze finite execution traces. LTL [Pnueli, 1977] is a well-accepted and established logic used for specifying properties of infinite traces, however, as referred, in RV, the goal is to check LTL properties given *finite prefixes* of infinite traces. As such, we will now give a description of two LTL-based specifications for finite traces.

ptLTL [Laroussinie et al., 2002] was proposed to extend the LTL with *past operators*. The principle of this logic is rather intuitive: something in the present implies that something happened in the past. ptLTL is a temporal logic where future-time modalities – F ("sometime in the future"), G ("always in the future"), U ("until"), and X ("next") – are complemented with their past-time counterparts – P or F^{-1} ("once in the past"), H or G^{-1} ("always in the past"), S or U^{-1} ("since"), and X^{-1} ("previous") – respectively. There is a duality between Past-time and Future-time logics, however, Gabbay [Gabbay, 1987] has proved that any linear-time temporal property expressed using past-time modalities can be translated into an equivalent (when evaluated at the beginning of the path), pure future formula. Actually, ptLTL is not more expressive than LTL, but it is more succinct than LTL. Gabbay also argues that this result also extends to other temporal logics, such as CTL* with past, μ-calculus with past, etc.

LTL_3, introduced by Bauer et al. [Bauer et al., 2011] is a logic which shares the syntax with LTL but deviates in its semantics for finite traces. The idea was to implement three truth values – $\top$ (*true*), $\bot$ (*false*), ? (*inconclusive*) – for the logic formulas. More precisely, given a finite word u and an LTL_3 formula φ, the interpretation of u is defined, according to [Bauer et al., 2011], as follows:

- if there is no continuation of u satisfying φ, the value of φ is *false*;
- if every continuation of u satisfies φ, the value of φ is *true*; and
- if true or false values cannot be determined, the value of φ is *inconclusive*.

Havelund and Rosu [Havelund and Rosu, 2002] propose a monitor synthesis algorithm for ptLTL formulas. The generated monitor tests whether the ptLTL formula is satisfied by a finite trace of events given as input and executed in linear time – depending on the

ptLTL formula size as well as the memory consumption. The synthesis process is basically a pretty-print, which is a direct conversion from the logic formula to the target programming language Java. The authors also suggest optimizations for the synthesis algorithm, which is part of PaX, and argue that it generates efficient monitors.

Bauer et al. [Bauer et al., 2011] have developed an algorithm for generating efficient monitors for discrete-time properties. Their approach only considers monitoring properties that are specified in LTL_3 or in TLTL with three truth values. They describe how *finite state machine*s (FSMs) with three output symbols are generated from LTL_3 formulas. The generated automaton reads finite traces and yields their three-valued semantics. Thus, monitors for three-valued formulas classify prefixes as being good ($\top$), bad ($\bot$), or neither good nor bad (?). Standard minimization techniques for FSMs can be applied to obtain a unique FSM that is optimal with respect to its number of states. The authors designed LTL_3 to specifically match the needs arising in RV.

Comparing both previous solutions, there are two important differences to note:

1. Bauer et al.'s solution uses LTL with three truth values instead of Havelund and Rosu's solution that uses ptLTL, and
2. Bauer et al.'s solution generates FSMs from LTL_3 formulas instead of Havelund and Rosu's solution that applies a direct conversion from ptLTL semantics to the program code (in this case, the Java programming language).

Two techniques that are less used but are related to the topic of this thesis. The Anna (ANNotated Ada) specification language was introduced in [Sankar and Mandal, 1993], including the synthesis monitor algorithm named Anna consistency checking system (Anna CCS). This outdated approach consists in the construction of a high-level specification language for concurrency monitoring. It is suitable to monitor the critical aspects of the system's behavior continuously along its execution. Anna is based on first order logic and its syntax is an extension of the Ada syntax. Anna CCS provides the capability to distribute the monitoring of specifications on multi-processor hardware platforms to meet practical time constraints. However, this approach assumes that the program under monitoring is sequentially executed. LOLA [D'Angelo et al., 2005] is also a specification language and an algorithm for the online and offline monitoring of synchronous systems, which include circuits and embedded systems. Even being a functional language over finite streams, the initial proposal does not contemplate support for runtime monitoring of synchronous systems using more than one clock, neither asynchronous systems. Due to that several streams acquired with different clocks cannot be used.

2.3.1 Runtime Monitoring of RTS

So far, not many approaches for RV of real-time properties have been proposed. In the following, three real-time monitoring approaches are described.

Temporal Rover [Drusinsky, 2000] is appropriate for monitoring of hard real-time systems due to the temporal constraints being specified in MTL in spite of the monitoring software being closed, therefore we are not able to undertansd how it is designed. Temporal Rover is a commercial RV tool based on future time metric temporal logic. It allows programmers to insert formal specification in programs via annotations, from which monitors are generated. An Automatic Test Generation (ATG) component is also provided to generate test sequences from logic specifications. Temporal Rover and its successor, DB Rover, support both inline and offline monitoring. However, they also have their specification formalisms hardwired and are tightly bound to Java. [Alves et al., 2011] presents the results of a formal computer-aided validation and verification of critical time-constrained requirements of the Brazilian Satellite Launcher flight software based on Temporal Rover.

In [Auguston and Trakhtenbrot, 2008] the authors present an approach for the dynamic analysis of reactive systems via RV of code generated from Statechart [Harel and Naamad, 1996] models and verified by the Statemate approach [Auguston and Trakhtenbrot, 2008]. The approach is based on the automatic synthesis of monitoring statecharts from formulas that specify the system's temporal and real-time properties in a proposed assertion language. The promising advantage of this approach is in its ability to analyze real-world models (with attributes reflecting the various design decisions) in the system's realistic environment. This capability is beyond the scope of model checking tools.

Bauer et al. have developed an algorithm for generating efficient monitors from TLTL for real-time systems [Bauer et al., 2011]. The authors introduce the notion of TLTL with three truth values, denoted TLTL_3. This basic notion is interesting and adequate for RV, since the complete set of traces is not available and the RV requires that the specification is evaluated increasingly. This approach employs so-called *event-clock automata* (ECA) for monitoring of TLTL_3 formulas. Moreover, Bauer et al. introduce the *symbolic timed runs* and show their benefits for checking specifications efficiently, avoiding a possible but generally expensive translation of ECA to predicting-free timed automata. Yet, without considering counting time explicitly.

2.3.2 Related Work

The last two decades have witnessed an immense increase in research activities in the

area of static analysis [Nielson et al., 1999, Almeida et al., 2011, Tschannen et al., 2011], where numerous theories and methods have been developed to verify both sequential and concurrent programs [Apt et al., 2009]. However, techniques such as model checking [Baier and Katoen, 2008, Clarke et al., 1999] and theorem proving [Harrison, 2009] proved to be hard, expensive and non intuitive for the common programmer (i.e., many times unusable [Tschannen et al., 2011]). Moreover, the trend towards increasing size and complexity of software in real-time systems promises to make their static verification very challenging in the foreseeable future [Zhu et al., 2009]. The exploration of other techniques, such as dynamic verification, is necessary in order to decrease the burden of program verification, either in alternative or as complement to static methods [Leucker and Schallhart, 2009, Falcone, 2010]. A recent trend in program verification is the use of runtime checking to complement the property verification of sequential and concurrent systems [Tschannen et al., 2011, Zee et al., 2007].

In this section, we will review some approaches to monitoring based on aspect-oriented programming, rule-based languages, and hardware monitoring.

Aspect-Oriented Programming Languages. Aspect-oriented programming is a recent paradigm to organize the entities according to aspects, which has proved to be adequate/useful for monitoring calls instrumentation. Aspect-oriented programming has been increasingly adopted in different programming languages, e.g., AspectJ (an aspect-oriented extension of Java language), AspectC++ (an aspect-oriented extension of C and C++ languages), and recently Ada 2012 [Barnes, 2012]. Building on these AOP languages, numerous extensions have been proposed to provide domain-specific features for AOP. Among these extensions, Tracematches [Allan et al., 2005] and J-LO [Bodden, 2004] support history(trace)-based aspects for Java.

Tracematches enables the programmer to trigger the execution of certain block of code by specifying a parametric regular pattern of events in a computation trace, where the events are defined over entry/exit of AspectJ pointcuts. When the pattern is matched during the execution, the associated code will be executed.

J-LO is a tool for runtime-checking temporal assertions. These temporal assertions are specified using parametric linear temporal logic (LTL) and the syntax adopted in J-LO is similar to Tracematches except that the properties are specified in a different formalism. J-LO also uses the same parametricity semantics as Tracematches. J-LO mainly focuses on checking properties at runtime rather than providing programming support. In J-LO, the temporal assertions are inserted into Java files as annotations that are then compiled into runtime checks. Both Tracematches and J-LO support parametric events, i.e., free

variables can be used in the specified properties and will be bound to specific values at runtime for matching events.

Rule-based Languages. Eagle [Barringer et al., 2004a], RuleR [Barringer et al., 2007], and PQL [Martin et al., 2005] are general specification languages which encompass monitoring algorithms. Such specification formalisms allow for complex property specification with parameter bindings. Eagle and RuleR are based on fixed-point logics and rewrite rules, while PQL is based on SQL relational queries. PQL allows programmers to express design rules that deal with sequences of events associated with a set of related objects. These schemes tackle the definition of specification language with the support of data binding among many other features, which makes the languages somewhat confusing and probably inefficient for monitor generation.

Program Trace Query Language (PTQL) [Goldsmith et al., 2005] is a language based on SQL-like relational queries over program traces. The current PTQL compiler, Partiqle, instruments Java programs to execute the relational queries on the fly. PTQL events are timestamped, and the timestamps can be explicitly used in queries. PTQL can be arbitrarily complex in the worst cases but, in average, it has an acceptable overhead. PTQL properties are globally scoped and their running mode is inline, as the predecessor PQL. PTQL provides no support for recovery, its main use being to detect errors. PTQL has static and dynamic tools. The static analysis conservatively looks for potential matches for queries and is useful to reduce the number of dynamic checks. The dynamic analyzer checks the runtime behavior and can perform user-defined actions when matches are found.

Attempts at monitoring hardware. BusMOP [Pellizzoni et al., 2008] is an outline hardware monitoring solution proposed to plug a monitor into a peripheral bus. The peripheral behavior is monitored by hardware, within which the read and write transactions are examined on the bus without runtime overhead on the system.

The PSL to Verilog compiler, P2V [Lu and Forin, 2008], is an attempt to perform runtime monitoring of formal properties in hardware. P2V is similar to BusMOP in that monitors are implemented in hardware rather than software, and that both approaches thus have no runtime overhead on the CPU. P2V, however, is more similar to the above approaches in that it is designed for monitoring actual programs rather than peripheral devices. Also it requires a dynamically extensible soft-core processor implemented on an FPGA, while the BusMOP approach can potentially be applied to any COTS communication architecture. Furthermore, P2V uses hardwired logic (PSL) while BusMOP allows for the use of different formalisms.

2.3.2.1 Frameworks

MOP [Meredith et al., 2011], RV [Meredith and Roşu, 2010], MaC [Kim et al., 2004], PathExplorer (PaX) [Havelund and Rosu, 2001], Eagle [Barringer et al., 2004b], RuleR [Barringer et al., 2010], and RMOR [Havelund, 2008] are RV frameworks for *logic*, *extended regular expressions* (ERE), *context-free grammar* (CFG), *assertion*-based monitoring, within which specific tools for Java (and C) – Java-MOP [Jin et al., 2012], RV-Monitor/RV-Predict, Java-MaC, Java PathExplorer, Hawk [d'Amorim and Havelund, 2005], and RMOR, respectively – are implemented. The summary of the specification languages of such platforms, which support outline monitoring, is the following

- MOP supports *extended regular expressionss* (EREs), *Java modeling language* (JML), and several variants of LTL;
- RV uses five different specification formalisms, namely FSMs, EREs, CFGs, *past-time linear temporal logic* (ptLTL), and *future-time linear temporal logic* (FTLTL);
- MaC uses a specialized language based on interval temporal logic;
- JPaX just supports LTL;
- Eagle adopts a first order fixed-point LTL with a chop operator;
- RuleR solves some performance issues of Eagle and adopts a fixed-point *propositional temporal logic* (PTL); and
- RMOR supports LTL and a graphical state machine language RCAT.

MOP, a monitor oriented programming framework, can be seen as having evolved from JPaX with the idea that the specification and implementation together form a system. The MOP approach supports inline, outline, and offline monitoring; it allows to define new formalisms to extend the MOP framework; it generates monitors from annotated code as plain Java code; and it adapts easily to new languages (as the authors argue). MTL currently is not supported by MOP, neither is any other real-time logic. The RV system [Meredith and Roşu, 2010], a commercial-grade successor of MOP, is based on the success of the MOP system and on a vastly expanded version of the jPredictor System [Chen et al., 2008]. MaC [Sokolsky et al., 2006, Sammapun et al., 2007] and JPaX integrate monitors via Java bytecode instrumentation, making them difficult to port to other languages. MaC also supports statistical runtime checking. Eagle attempts to build a logic that is powerful enough to subsume most existing specification logics. The Eagle logic with a chop operator allows to model sequential composition. Although quite expressive, it does not yield efficient monitors, so RuleR attempts to address those inefficiencies [Goodloe and Pike, 2010]. A monitor is expressed as a collection of logic rules specified in propositional

temporal logic, as a FSM, or CFG. The RMOR platform monitors C programs specifying both safety and bounded liveness properties that can be expressed as FSMs, and observes events recorded in an execution trace.

These platforms are only suited for runtime monitoring or even RV of concurrent systems. As such, they cannot be used for real-time systems since only temporal constraints are ensured, and as it is well known that real-time systems are mainly characterized by their dependence on timing (or timed) constraints.

2.3.2.2 RV vs. static verification and testing techniques

Due to the increasing importance of contextualizing verification techniques in the sense of knowing their potential and fragilities, a comparison between RV and three well-known techniques (deductive reasoning, model checking and testing) is made in the following paragraphs. These techniques can be characterized in terms of *scalability*, types of *properties* and *coverage*.

Model Checking. RV shares many similarities with model checking and, roughly speaking, this technique can be seen as complementary to model checking (i.e., runtime verification reduces verification issues, which are undecidable, but also reduces the coverage). Nevertheless, and according to [Leucker and Schallhart, 2009], there are important differences to consider:

1. In RV, only one execution of a given system is checked to answer, in execution time or after the execution (inline monitoring and outline monitoring, respectively), whether it satisfies a given correctness property φ. This corresponds to knowing whether the execution trace satisfies the property φ, i.e., the word acceptance problem. In contrast, model checking deals with the language inclusion problem. As is well-known, the word problem is of far lower complexity than the inclusion problem [Alur and Dill, 1994].

2. RV considers finite traces, since all executions are necessarily finite, whereas model checking deals with infinite traces.

3. RV, especially when dealing with online monitoring, considers finite executions of increasing size. For this, a monitor should be designed to consider executions in an incremental fashion. In contrast, model checking deals with a complete model which allows considering arbitrary positions of a trace.

From an application point of view, there are also important differences between RV and model checking: RV deals only with observed executions. Thus it is applicable to black-box systems for which no system model is at hand. In model checking, however, a precise description of the system to check is mandatory as, before actually running the system, all possible executions must be checked. Furthermore, model checking suffers from the well-known *state explosion problem*, which refers to the fact that analyzing all executions of a system is typically carried out by generating the whole state space, which often becomes unfeasibly huge. Considering a simple run, on the other hand, most applications of RV are not practically limited by their memory requirements, since the necessary history information, although potentially unbounded, is usually fairly small.

Model checking is characterized by a lower scalability (due to the state explosion problem), a lower properties coverage (several properties cannot be checked, e.g., explicit time properties, especially when dealing with TPTL and MTL), and higher coverage of the model (e.g., a property φ holds for all possible paths of the model). However, verification using model checking is only as good as the model of the system.

Deductive Reasoning. Logical deduction is clearly one of the most used techniques in software verification; however, it is also one of the most difficult to apply. Deductive proof construction and RV are two distinct techniques, clearly without similarities. They differ in the following points:

1. Deductive proofs are much more time-consuming than a push button operation such as RV in the sense of utilization perspective. Deductive reasoning requires that well known deductive techniques and tactics are used. Moreover, all RV tools works in an automatic fashion.

2. RV has lower coverage than deductive proofs. The latter technique is general and comprehensive. In contrast, RV only verifies concrete past executions which cannot be extended or generalized.

3. Deductive proofs are exact and rigorous, no more verification efforts are required after a finite set of steps are found.

Testing. RV has similarities with testing since neither of the techniques considers each possible execution of a system, but just a single or a finite subset, indicating that their coverage is usually incomplete.

There are two testing schemes, namely *suite-based* testing and *oracle-based* testing, that can be used [Hamlet, 2010]. Typically, a test suite is formed by a finite set of finite input-

output sequences. Test-case execution is then the act of checking whether the output of a system agrees with the predicted one, after giving the input sequence to the system under test. However, oracle-based testing, a closer approach to RV, composes a test suite which is only formed by input sequences. To anticipate the output results for testing, a so-called *test oracle* has to be designed and coupled to the system under test. This oracle observes the system under test and checks a number of properties (e.g., by unit tests [4]) in an automatic way. In contrast, RV is identical in the sense that the monitor is coupled to the system and instead of testing it, monitors whether the properties are satisfied or violated. An alternative way to compare both techniques, according to [Bauer et al., 2011], is:

1. RV generates monitors from high-level specifications rather than a handmade construction of a test oracle.

2. RV does not consider the supply of a suitable set of input sequences to exhaustively test a system.

This technique is the most widely used in industry due mainly to its greater scalability, in spite of lower coverage and the uncertainty in test oracle development.

Summary

Indispensable topics and respective related works have been summarized and merged in this chapter as the background required to read this thesis. We have recapped the importance of duration properties for proving correct real-time systems' behavior with the conclusion that there is a huge gap of RV frameworks ready to deal with explicit real-time systems properties. Proper languages for describing hard real-time systems properties have been surveyed as well, including a diverse number of properties. One important language is MTL-$\int$, which, being more expressive than duration calculus, may originate further issues that need to be dealt statically (we will continue exploring it in the next chapter). We have also introduced lambda calculus as basic and elegant theory for constructing new synthesis algorithms.

[4] Consists of testing certain areas of the source-code by providing different inputs for such blocks of code (e.g., functions) and comparing it with the desired outcome.

Chapter 3

RV with RMTL-$\int$

RV is concerned with the problem of generating monitors from formal specifications, and adding these monitors into the target code as a safety-net that is able to detect abnormal behaviors and, possibly, respond to them via the release of counter-measures. Providing an expressive formal language that fits the timing requirements of real-time systems is the main objective of this chapter.

A fragment of MTL-$\int$ is presented as an intuitive tool to carry out RV of hard real-time systems. We begin by the *specification language* and then introduce the notions of *inequality translation* using $\text{FOL}_{\mathbb{R}}$ in order to simplify *restricted metric temporal logic with durations* (RMTL-$\int$) formulas. In the remaining part of the chapter, we present the correctness result of the inequality translation algorithm, and we conclude by describing the synthesis algorithms for static and dynamic verification purposes.

3.1 The specification Language RMTL-$\int$

To overcome the undecidability results of MTL-$\int$, we will apply restrictions on its definition. RMTL-$\int$ is a syntactically and semantically restricted fragment of MTL-$\int$; the syntactic restrictions over MTL-$\int$ include the use of *bounded formulas*, of a single relation $<$ over the real numbers, the restriction of the n-ary function terms to use one of the $+$ or $\times$ operators, and a restriction of α constants to the set of rationals $\mathbb{Q}$. Tarski's theorem [Tarski, 1995] states that the first-order theory of reals with $+$, $\times$, and $<$ allows for quantifiers to be eliminated. Algorithmic quantifier elimination leads to decidability, assuming that the truth values of formulas involving only constants (without free variables and bound variables) can be computed.

The semantic restrictions on the other hand include the conversion of the *continuous* semantics of MTL-$\int$ into an *interval-based* semantics, where models are *timed interval sequences* and formulas are evaluated in a given logical environment at a time $t \in \mathbb{R}_{\geq 0}$.

Definition 6 (RMTL-$\int$ formulae)**.** Let $\mathcal{P}$ be a set of propositions and $\mathcal{V}$ a set of logical variables. The syntax of RMTL-$\int$ terms η and formulas φ is defined inductively as follows:

$$\eta \ ::= \ \alpha \mid x \mid \eta_1 \circ \eta_2 \mid \int^{\eta} \varphi$$
$$\varphi \ ::= \ \mathit{true} \mid p \mid \eta_1 < \eta_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \varphi_1 \ \mathrm{U}_{\sim\gamma} \ \varphi_2 \mid \varphi_1 \ \mathrm{S}_{\sim\gamma} \ \varphi_2 \mid \exists x \, \varphi$$

where: $\alpha \in \mathbb{R}$, $x \in \mathcal{V}$ is a logical variable, the operators $\circ \in \{+, \times\}$ are used for the sum and multiplication of terms, $\int^{\eta} \varphi$ is the duration of the formula φ in the interval $[0, \eta]$; $p \in \mathcal{P}$ is an atomic proposition, $<$ is the relation *less than* on terms, U and S are temporal operators, with $\sim \in \{<, =\}$ and $\gamma \in \mathbb{R}_{\geq 0}$.

We will use the following classic shorthands: $\varphi \wedge \psi$ for $\neg(\neg\varphi \vee \neg\psi)$, $\varphi \rightarrow \psi$ for $\neg\varphi \vee \psi$, $\Diamond_{\sim\gamma} \varphi$ for $\mathit{true} \, \mathrm{U}_{\sim\gamma} \, \varphi$, and $\Box_{\sim\gamma} \varphi$ for $\neg(\mathit{true} \, \mathrm{U}_{\sim\gamma} \, \neg\varphi)$. We will denote by Φ the set of RMTL-$\int$ formulas. Furthermore, we will use $\circ \in \{+, \times\}$ and $\sim \in \{<, =\}$ to range over operators.

A *timed state sequence* κ is an infinite sequence of the form

$$(p_0, [i_0, {i'}_0[), (p_1, [i_1, {i'}_1[\ldots,$$

where $p_j \in \mathcal{P}$, $i'_j = i_{j+1}$ and $i_j, i'_j \in \mathbb{R}_{\geq 0}$ such that $i_j < i'_j$ and $j \geq 0$. Let $\kappa(t)$ be defined as $\{p_j\}$ if there exists a tuple $(p_j, [i_j, {i'}_j[)$ such that $t \in [i_j, {i'}_j[$, and as $\emptyset$ otherwise. Note that there exists at most one such tuple.

A *logical environment* is any function $\upsilon : \mathcal{V} \rightarrow \mathbb{R}_{\geq 0}$. For any $x \in \mathcal{V}$, $r \in \mathbb{R}$, and logical environment υ, we will denote by $\upsilon[x \mapsto r]$ the logical environment that maps x to r and every other variable y to $\upsilon(y)$.

Definition 7 (RMTL-$\int$ semantics)**.** The truth value of a formula φ will be defined relative to a model (κ, υ, t) consisting of a timed state sequence k, a logical environment υ, and a time instant $t \in \mathbb{R}_{\geq 0}$. We will write $(\kappa, \upsilon, t) \models \varphi$ when φ is interpreted as true in the model (κ, υ, t). Terms and formulas will be interpreted in a mutually recursive way.

First of all, for each formula φ, timed state sequence k and logical environment υ, the auxiliary *indicator function* $1_{\varphi(\kappa,\upsilon)} : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ is defined as follows, making use of the satisfaction relation:

$$1_{\varphi(\kappa,\upsilon)}(t) = \begin{cases} 1 & \text{if } (\kappa, \upsilon, t) \models \varphi, \\ 0 & \text{otherwise.} \end{cases}$$

The value $\mathscr{T}[\![\eta]\!]_{(\kappa,\upsilon)\,t}$ of a term η relative to a model can then be defined. A *Riemann integral* [Gordon, 1994] of the function $1_{\varphi(\kappa,\upsilon)}$ is used for the case of a duration $\int^{\eta}\varphi$.

$$
\begin{aligned}
\mathscr{T}[\![\alpha]\!]_{(\kappa,\upsilon)\,t} &= \alpha \\
\mathscr{T}[\![x]\!]_{(\kappa,\upsilon)\,t} &= \upsilon(x) \\
\mathscr{T}[\![\eta_1 \circ \eta_2]\!]_{(\kappa,\upsilon)\,t} &= \mathscr{T}[\![\eta_1]\!]_{(\kappa,\upsilon)\,t} \circ \mathscr{T}[\![\eta_2]\!]_{(\kappa,\upsilon)\,t} \\
\mathscr{T}\left[\!\!\left[\int^{\eta}\varphi\right]\!\!\right]_{(\kappa,\upsilon)\,t} &= \begin{cases} \int_t^{t+\mathscr{T}[\![\eta]\!]_{(\kappa,\upsilon)t}} 1_{\varphi(\kappa,\upsilon)}(t_*)\; dt_* & \text{if } \mathscr{T}[\![\eta]\!]_{(\kappa,\upsilon)\,t} \geq 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

The satisfaction relation is defined inductively as follows:

$$
\begin{array}{lll}
(\kappa,\upsilon,t) \models \mathit{true} & & \\
(\kappa,\upsilon,t) \models p & \text{iff} & p \in \kappa(t) \\
(\kappa,\upsilon,t) \models \eta_1 < \eta_2 & \text{iff} & \mathscr{T}[\![\eta_1]\!]_{(\kappa,\upsilon)\,t} < \mathscr{T}[\![\eta_2]\!]_{(\kappa,\upsilon)\,t} \\
(\kappa,\upsilon,t) \models \varphi_1 \vee \varphi_2 & \text{iff} & (\kappa,\upsilon,t) \models \varphi_1 \text{ or } (\kappa,\upsilon,t) \models \varphi_2 \\
(\kappa,\upsilon,t) \models \neg\varphi & \text{iff} & (\kappa,\upsilon,t) \not\models \varphi \\
(\kappa,\upsilon,t) \models \varphi_1 \ \mathrm{U}_{\sim\gamma}\ \varphi_2 & \text{iff} & \text{there exists } t' \text{ such that } t \leq t' \sim t+\gamma \text{ and } (\kappa,\upsilon,t') \models \varphi_2, \\
 & & \text{and for all } t'' \text{such that } t < t'' < t',\ (\kappa,\upsilon,t'') \models \varphi_1 \\
(\kappa,\upsilon,t) \models \varphi_1 \ \mathrm{S}_{\sim\gamma}\ \varphi_2 & \text{iff} & \text{there exists } t' \text{ such that } t-\gamma \sim t' \leq t \text{ and } (\kappa,\upsilon,t') \models \varphi_2, \\
 & & \text{and for all } t'' \text{such that } t' < t'' < t,\ (\kappa,\upsilon,t'') \models \varphi_1 \\
(\kappa,\upsilon,t) \models \exists x\,\varphi & \text{iff} & \text{there exists a value } r \in \mathbb{R} \text{ such that } (\kappa,\upsilon[x \mapsto r],t) \models \varphi
\end{array}
$$

We will write $(\kappa,\upsilon) \models \varphi$ as shorthand for $(\kappa,\upsilon,0) \models \varphi$. Note that the semantics of the until operator is strict and non-matching. This implies that, in order to satisfy $\varphi_1\ \mathrm{U}_{\sim\gamma}\ \varphi_2$, the model is not required to satisfy φ_1.

An important property of our restriction is that RMTL-$\int$ satisfies by construction the Dirichlet condition implying the Riemann property [Lakhnech and Hooman, 1995, p.7]:

Lemma 1. *For any RMTL-$\int$ formula φ, timed state sequence κ, and logical environment υ, the indicator function $1_{\varphi(\kappa,\upsilon)}$ is Riemann integrable.*

Proof of Lemma 1. We proceed by contradiction on the claim that the function $1_{\varphi(\kappa,\upsilon)}$ has finitely many discontinuities. Let us consider the model (κ,υ,t) and a proposition *prop* such that $\mathit{prop} \in \kappa(t)$ for $t \in [0,1)$.

We consider the case when ϕ is equal to $\int^1 \mathit{prop} = 1-a$: from the semantic interpretation of the duration term, we have $\mathscr{T}\left[\!\!\left[\int^1 \mathit{prop}\right]\!\!\right]_{(\kappa,\upsilon)\,t} = 1-t$. Applying the substitution property of equality, we get $a + \int^1 \mathit{prop} = 1$. Since t is directly related to the variable a, when the timed state sequence κ has finite length, from the semantic rules we can see that

if a has infinitely many discontinuities along t then $1_{\phi(\kappa,\upsilon)}$ also contains infinitely many discontinuities. Considering the above relation between t and the logic variable a $(t = a)$, introducing infinitely many discontinuities in t means that we can extend the formula ϕ to introduce finitely many discontinuities in a. Now, from a close examination of the semantics of the logic, we have that a can be constrained only by polynomial inequalities. Infinite discontinuities on polynomial inequalities are not obtainable.

We also need to consider the case when Boolean operators are applied to polynomial inequalities. In order to obtain an infinite number of discontinuities we would need an infinite number of Boolean operators and then an infinite formula. Since any formula needs to be finite to be satisfiable, then this contradicts the claim.

We skip the proof for the remaining cases, since no more relations between t and logic variables can be allowed semantically, other than those originating in duration terms in certain circumstances. To conclude the proof, we have that no infinitely many discontinuities exist, and then the Dirichlet condition holds, which implies that the indicator function $1_{\varphi(\kappa,\upsilon)}$ is Riemann integrable. □

Example 4 (Application of Durations)**.** *Let us now consider an example using a duration term concerning the evolution of a real-time system formed by tasks depending entirely on the occurrence of events, the evaluation of the propositions is performed over these events, and all the tasks have an associated fixed set of events. Let ϕ_m be a formula that specifies the periodic release of a renewal event for a timed resource in the system, and let ψ_m be a formula specifying every event triggered by tasks belonging to that resource. To monitor utilization and the release of timed resources, we employ the formula,*

$$\Box_{<\upsilon}\ \phi_m \rightarrow \int^{t} \psi_m \leq \beta,$$

where υ is arbitrarily large, t is the budget renewal period, and β is the allowed budget (i.e. the execution time of tasks belonging to the timed resource). Let us consider two finite sequences κ_1 and κ_2, such that κ_1 is a subsequence of κ_2, and an arbitrary formula ϕ. In the two-valued setting, incremental evaluation over t is inconsistent with respect to the sequence, since we could have $(\kappa_1, \upsilon, 0) \not\models \phi$ and $(\kappa_2, \upsilon, 10) \models \phi$ due to lack of sequence symbols in κ_1.

A different solution will be presented in the next section where the *unknown* truth-value is an option.

3.2 Three-valued Extension of RMTL-$\int$

The three-valued logic extension of RMTL-$\int$, which we will call *three-valued restricted metric temporal logic with durations* (RMTL-$\int_3$), is syntactically defined as before, but contains two new terms. These terms allow for variables to be maximized and minimized in certain intervals, subject to a constraint given as a formula. The terms must be introduced here due to the situation in which no minimum or maximum exists (the formula is not satisfied in the interval), since we need to define an infeasible value instead of assigning a real number to these terms. The language of terms of RMTL-$\int_3$ is defined as follows:

$$\eta \ ::= \ \alpha \mid x \mid \min_x \varphi \mid \max_x \varphi \mid \eta_1 \circ \eta_2 \mid \int^{\eta} \varphi$$

where $\min_x \varphi$ and $\max_x \varphi$, are respectively, the minimum and maximum of a formula with respect to the logical variable x. All other formulas and terms are as in RMTL-$\int$. We will denote by Φ^3 the set of RMTL-$\int_3$ formulas, and by Γ the set of RMTL-$\int_3$ terms.

Definition 8 (RMTL-$\int_3$ Semantics)**.** The truth value of a formula φ will again be defined relative to a model (κ, υ, t) consisting of a timed state sequence k, a logical environment υ and a time instant $t \in \mathbb{R}_{\geq 0}$, and will now be one of the 3-values $\{\mathsf{tt}, \mathsf{ff}, \bot\}$. We will write $[\![\varphi]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$ when φ is interpreted as true in the model (κ, υ, t), $[\![\varphi]\!]_{3(\kappa,\upsilon,t)} = \mathsf{ff}$ when φ is interpreted as false in the model (κ, υ, t), and $[\![\varphi]\!]_{3(\kappa,\upsilon,t)} = \bot$ otherwise. The auxiliary indicator function $1_{\varphi(\kappa,\upsilon)} : \mathbb{R}_{\geq 0} \to \{-1, 0, 1\}$ is defined as follows:

$$1_{\varphi(\kappa,\upsilon)}(t) = \begin{cases} 1 & \text{if } [\![\varphi]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}, \\ 0 & \text{if } [\![\varphi]\!]_{3(\kappa,\upsilon,t)} = \mathsf{ff}, \\ -1 & \text{if } [\![\varphi]\!]_{3(\kappa,\upsilon,t)} = \bot \end{cases}$$

The interpretation of the term η will be given by $\mathscr{T}[\![\eta]\!]_{3(\kappa,\upsilon)}\, t \in \mathbb{R} \cup \{\bot_{\mathbb{R}}\}$, as defined by the following rules. Whenever $\mathscr{T}[\![\eta]\!]_{3(\kappa,\upsilon)}\, t = \bot_{\mathbb{R}}$, this means that the term η is *infeasible*.

Rigid terms:

- $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)}\, t$ is defined as α if $\eta_1 = \alpha$, and as $\upsilon(x)$ if $\eta_1 = x$

Minimum and Maximum terms:

- If $\eta_1 = \min_x \varphi$, then $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)}\, t$ is defined as:

$$\begin{cases} \min \mathfrak{m} & \text{if } \mathfrak{m} \neq \emptyset \text{ and for all } y \text{ such that } y < \min m, [\![\varphi]\!]_{3(\kappa,\upsilon[x\mapsto y],t)} \neq \bot \\ \bot_{\mathbb{R}} & \text{otherwise} \end{cases}$$

  where $\mathfrak{m} = \{r \mid [\![\varphi]\!]_{3(\kappa,\upsilon[x\mapsto r],t)} = \mathsf{tt}\}$.

- If $\eta_1 = \max\limits_{x} \varphi$, then $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t}$ is defined as:

$$\begin{cases} \max \mathfrak{n} & \text{if } \mathfrak{n} \neq \emptyset \text{ and for all } y \text{ such that } \max n < y, [\![\varphi]\!]_{3(\kappa,\upsilon[x\mapsto y],t)} \neq \bot \\ \bot_{\mathbb{R}} & \text{otherwise} \end{cases}$$

  where $\mathfrak{n} = \{r \mid [\![\varphi]\!]_{3(\kappa,\upsilon[x\mapsto r],t)} = \mathsf{tt}\}$.

Duration term:

- If $\eta_1 = \int^{\eta_2} \phi$, then $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t}$ is defined as:

$$\begin{cases} \int_t^{t+\mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t}} 1_{\phi(\kappa,\upsilon)}(t')\ dt' & \text{if } \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} \geq 0 \text{ and for all } t'' \in [t, t+\mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t}], \\ & \quad 1_{\phi(\kappa,\upsilon)}(t'') \in \{0,1\} \\ \bot_{\mathbb{R}} & \text{otherwise} \end{cases}$$

Binary terms:

- If $\eta_1 = \eta_2 + \eta_3$, then $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t}$ is defined as:

$$\begin{cases} \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} + \mathscr{T}[\![\eta_3]\!]_{3(\kappa,\upsilon)\,t} & \text{if } \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t}, \mathscr{T}[\![\eta_3]\!]_{3(\kappa,\upsilon)\,t} \in \mathbb{R} \\ \bot_{\mathbb{R}} & \text{otherwise} \end{cases}$$

- If $\eta_1 = \eta_2 \times \eta_3$, then $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t}$ is defined as:

$$\begin{cases} \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} \times \mathscr{T}[\![\eta_3]\!]_{3(\kappa,\upsilon)\,t} & \text{if } \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t}, \mathscr{T}[\![\eta_3]\!]_{3(\kappa,\upsilon)\,t} \in \mathbb{R} \\ \bot_{\mathbb{R}} & \text{otherwise} \end{cases}$$

Turning to the interpretation of formulas, we define $[\![\varphi]\!]_{3(\kappa,\upsilon,t)}$ to be one of the three values in $\{\mathsf{tt}, \mathsf{ff}, \bot\}$, according to the following rules.

Basic formulae:

- If ϕ is p, then $[\![\phi]\!]_{3(\kappa,\upsilon,t)}$ is tt if $p \in \kappa(t)$, ff if $p \notin \kappa(t)$ and $\kappa(t) \neq \emptyset$, and $\bot$ if $\kappa(t) = \emptyset$.

Relation operator:

- If ϕ is $\eta_1 < \eta_2$, then $[\![\phi]\!]_{3(\kappa,\upsilon,t)}$ is defined as:

$$\begin{cases} \mathsf{tt} & \text{if } \mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t}, \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} \in \mathbb{R}, \text{ and } \mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} < \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} \\ \mathsf{ff} & \text{if } \mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t}, \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} \in \mathbb{R}, \text{ and } \mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} \geq \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} \\ \bot & \text{if } \mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} = \bot_{\mathbb{R}} \text{ or } \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} = \bot_{\mathbb{R}} \end{cases}$$

Boolean operators:

- If ϕ is $\neg\varphi$, then $[\![\phi]\!]_{3(\kappa,\upsilon,t)}$ is tt if $[\![\varphi]\!]_{3(\kappa,\upsilon,t)} = \mathsf{ff}$, ff if $[\![\varphi]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$, and $\bot$ otherwise.

- If ϕ is $\varphi_1 \vee \varphi_2$, then $[\![\phi]\!]_{3(\kappa,\upsilon,t)}$ is tt if $[\![\varphi_1]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$ or $[\![\varphi_2]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$, ff if $[\![\varphi_1]\!]_{3(\kappa,\upsilon,t)} = \mathsf{ff}$ and $[\![\varphi_2]\!]_{3(\kappa,\upsilon,t)} = \mathsf{ff}$, and $\bot$ otherwise.

Temporal Operators:

- If ϕ is $\varphi_1 \ \mathrm{U}_{\sim\gamma} \ \varphi_2$, then $[\![\phi]\!]_{3(\kappa,\upsilon,t)}$ is defined as:

$$\begin{cases} \mathsf{tt} & \text{if there exists } t' \text{ such that } t \leq t' \sim t+\gamma,\ [\![\varphi_2]\!]_{3(\kappa,\upsilon,t')} = \mathsf{tt} \text{ and} \\ & \quad \text{for all } t'',\ t < t'' < t',\ [\![\varphi_1]\!]_{3(\kappa,\upsilon,t'')} = \mathsf{tt} \\ \mathsf{ff} & \text{if for all } t',\ t \leq t' \sim t+\gamma, \\ & \quad [\![\varphi_2]\!]_{3(\kappa,\upsilon,t')} \neq \mathsf{ff} \text{ implies that} \\ & \qquad \text{there exists } t'' \text{ such that } t < t'' < t', [\![\varphi_1]\!]_{3(\kappa,\upsilon,t'')} = \mathsf{ff} \text{ and} \\ & \quad [\![\varphi_2]\!]_{3(\kappa,\upsilon,t')} = \mathsf{ff} \text{ implies that there exists no } t'' \text{ such that } t < t'' < t' \text{ or} \\ & \qquad \text{there exists } t'' \text{ such that } t < t'' < t', [\![\varphi_1]\!]_{3(\kappa,\upsilon,t'')} = \mathsf{ff} \\ \bot & \text{otherwise} \end{cases}$$

- If ϕ is $\varphi_1 \ \mathrm{S}_{\sim\gamma} \ \varphi_2$, then $[\![\phi]\!]_{3(\kappa,\upsilon,t)}$ is defined as:

$$\begin{cases} \mathsf{tt} & \text{if there exists } t' \text{ such that } t-\gamma \sim t' \leq t,\ [\![\varphi_2]\!]_{3(\kappa,\upsilon,t')} = \mathsf{tt} \text{ and} \\ & \quad \text{for all } t'',\ t' < t'' < t, [\![\varphi_1]\!]_{3(\kappa,\upsilon,t'')} = \mathsf{tt} \\ \mathsf{ff} & \text{if for all } t',\ t-\gamma \sim t' \leq t, \\ & \quad [\![\varphi_2]\!]_{3(\kappa,\upsilon,t')} \neq \mathsf{ff} \text{ implies that} \\ & \qquad \text{there exists } t'' \text{ such that } t' < t'' < t,\ [\![\varphi_1]\!]_{3(\kappa,\upsilon,t'')} = \mathsf{ff} \text{ and} \\ & \quad [\![\varphi_2]\!]_{3(\kappa,\upsilon,t')} = \mathsf{ff} \text{ implies that there exists no } t'' \text{ such that } t' < t'' < t \text{ or} \\ & \qquad \text{there exists } t'' \text{ such that } t' < t'' < t,\ [\![\varphi_1]\!]_{3(\kappa,\upsilon,t'')} = \mathsf{ff} \\ \bot & \text{otherwise} \end{cases}$$

Existential operator:

- If ϕ is $\exists x \ \varphi$, then $[\![\phi]\!]_{3(\kappa,\upsilon,t)}$ is defined as:

$$\begin{cases} \mathsf{tt} & \text{if there exists a value } r \in \mathbb{R} \text{ such that } [\![\varphi]\!]_{3(\kappa,\upsilon[x\mapsto r],t)} = \mathsf{tt} \\ \mathsf{ff} & \text{if for all } r \in \mathbb{R},\ [\![\varphi]\!]_{3(\kappa,\upsilon[x\mapsto r],t)} = \mathsf{ff} \\ \bot & \text{there exits } r \in \mathbb{R} \text{ such that } [\![\varphi]\!]_{3(\kappa,\upsilon[x\mapsto r],t)} = \bot \text{ and} \\ & \quad \text{there exists no } r \in \mathbb{R} \text{ such that } [\![\varphi]\!]_{3(\kappa,\upsilon[x\mapsto r],t)} = \mathsf{tt} \end{cases}$$

We will write $(\kappa,\upsilon,t) \models_3 \varphi$ when $[\![\varphi]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$, and $(\kappa,\upsilon,t) \not\models_3 \varphi$ when $[\![\varphi]\!]_{3(\kappa,\upsilon,t)} = \mathsf{ff}$. In what follows we will often write $\eta_1 = \eta_2$ for $\neg(\eta_1 < \eta_2) \wedge \neg(\eta_2 < \eta_1)$.

Preservation of RMTL-$\int$ Semantics. An immediate motivation for (the choice of) defining a three-valued semantics for our logic fragment comes from the nature of runtime verification, which evaluates timed sequences where it is not possible to determine a definitive true or false value without analyzing the complete trace. For instance, considering a

prefix $\varkappa_p$ of a timed sequence $\varkappa$, we have that the evaluation of the same formula in the models $(\varkappa, \upsilon, t)$ and $(\varkappa_p, \upsilon, t)$ produces different truth values. Classic semantics cannot provide a common truth value to make consistent incremental evaluations of the model, which is an important feature for RV.

The semantic preservation of both truth and falsity for the three-valued logic is defined using the following two relations: a partial relation $\prec \subseteq \{\mathsf{tt}, \mathsf{ff}, \bot\} \times \{\mathsf{tt}, \mathsf{ff}\}$ defined by $\mathsf{tt} \prec \mathsf{tt}$, $\mathsf{ff} \prec \mathsf{ff}$, and $\bot \prec \mathsf{ff}$; and a partial relation $\lhd \subseteq \mathbb{R} \cup \{\bot_{\mathbb{R}}\} \times \mathbb{R}$ defined by $\bot_{\mathbb{R}} \lhd 0$, and $m \lhd m$, for all $m \in \mathbb{R}$, which gives a distinct treatment to duration terms that evaluate to 0 in the 2-valued semantics.

We will now formulate two auxiliary results required to prove the semantic preservation of RMTL-$\int$ in RMTL-$\int_3$. From a close examination of the minimum and maximum term semantics, we have that these terms are indeed quantified formulas, interpreted as a minimum or a maximum value that satisfies the quantification, or as $\bot_{\mathbb{R}}$ when this minimum or maximum is nonexistent. First of all we observe that the following axioms [Tarski, 1995, p. 205], where ϕ does not contain minimum and maximum terms, extend to our present setting:

A 1. $\eta_1 \circ \min\limits_x \phi < \eta_2 \iff (\forall y\ y < x \to \neg\phi[y/x]) \wedge \eta_1 \circ x < \eta_2 \wedge \phi.$

A 2. $\eta_1 \circ \max\limits_x \phi < \eta_2 \iff (\forall y\ y > x \to \neg\phi[y/x]) \wedge \eta_1 \circ x < \eta_2 \wedge \phi.$

A 3. $\int^{\eta_3} \phi_1 \circ \eta_1 \sim \eta_2 \iff x = \eta_3 \wedge \int^{x} \phi_1 \circ \eta_1 \sim \eta_2$

Axioms A1 and A2 indicate that a formula containing a minimum/maximum term is indeed a quantified formula constrained by the mim/max of the variable x. Axiom A3 replaces a formula containing a duration constrained in an interval by a duration term constrained by a logic variable. The meaning of $\phi \iff \psi$ is that $(\kappa, \upsilon, t) \models_3 \phi$ iff $(\kappa, \upsilon, t) \models_3 \psi$, for a model (κ, υ, t).

Lemma 2. *Let ϕ be a RMTL-$\int_3$ formula such that minimum and maximum terms only occur outside of the duration terms. Then, there exists an equivalent RMTL-$\int_3$ formula containing no occurrences of minimum and maximum terms.*

Proof. The proof follows by induction on the structure of the formula ϕ. We only present the case when ϕ is $\eta_1 < \eta_2$. We have to prove that there exists an equivalent form for the minimum and maximum terms for RMTL-$\int_3$ formulas. In particular, for all η_3 and η_4 and for any x and ϕ_1, the following holds

$$\eta_3 + \eta_4 \times \min_x \phi_1 < z \iff (\forall y\ y < x \to \neg\phi_1[y/x]) \wedge \eta_3 + \eta_4 \times x < z \wedge \phi_1.$$

Suppose η_1 is $\eta_3 + \eta_4 \times \min\limits_x \phi_1$ and η_2 is $\eta_5 + \eta_6 \times \min\limits_x \phi_2$. Assuming that $\eta_1 \neq z$ and $\eta_2 \neq z$, by the fourth axiom of the second axiomatization of Tarski [Tarski, 1995], we have that $\eta_1 < z \wedge z < \eta_2$, i.e.

$$\eta_3 + \eta_4 \times \min_x \phi_1 < z \wedge \neg \left(\eta_5 + \eta_6 \times \min_x \phi_2 \leq z \right).$$

Now, we have both inequalities in the same shape and we can consider the first one for continuing the proof (since the proof for the other inequality is similar). By axiom A1, we have $(\forall a\ a < x \rightarrow \neg\phi_1[a/x]) \wedge \eta_3 + \eta_4 \times x < z \wedge \phi_1$. By induction hypothesis we have

$$(\forall a\ a < x \rightarrow \neg\phi_1[a/x]) \wedge (\eta_6 + \eta_7 \times \min_y \phi_2) + (\eta_8 + \eta_9 \times \min_w \phi_3) \times x < z \wedge \phi_1.$$

Re-applying Axiom 1, we have that

$$\begin{aligned} &(\forall a\ a < x \rightarrow \neg\phi_1[a/x]) \wedge \\ &(\forall b\ b < y \rightarrow \neg\phi_2[b/y]) \wedge \\ &(\forall c\ c < w \rightarrow \neg\phi_3[c/w]) \wedge \\ &\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \\ &(\eta_6 + \eta_7 \times y) + (\eta_8 + \eta_9 \times w) \times x < z. \end{aligned}$$

Hence, the minimum terms vanish. We skip the case when $\eta_1 = z$, $\eta_2 \neq z$ and $\eta_1 \neq z$, $\eta_2 = z$; and also when the maximum term is employed (which makes use of axiom A2), since the proof is similar. □

From Lemma 2, we conclude that the minimum and maximum terms do not increase the RMTL-$\int_3$ expressiveness as they are indeed *syntatic sugar* that can be eliminated. We have not considered the situation when minimum and maximum terms occur in the scope of duration terms. For that we need to apply axiom A3 to replace the bound term of the duration, allowing for Lemma 2 to be further applied.

Now, given the result of Lemma 2, we will add the minimum and maximum terms to the syntax and semantics of RMTL-$\int$, since there is no difference from the expressiveness standpoint. Then, we will prove by mutual structural induction on the formula that the semantics is preserved. Let us define $\mathfrak{m} = \{r \mid [\![\varphi]\!]_{(\kappa,\upsilon[x\mapsto r],t)} = \mathtt{tt}\}$. The minimum term $\min\limits_x \varphi$ is semantically interpreted as an RMTL-$\int$ term as:

$$\mathscr{T}\left[\!\left[\min_x \varphi\right]\!\right]_{(\kappa,\upsilon)} t = \begin{cases} \min \mathfrak{m} & \text{if } \mathfrak{m} \neq \emptyset \\ 0 & \text{otherwise} \end{cases}.$$

The maximum term $\max\limits_x \varphi$ is semantically defined as:

$$\mathscr{T}\left[\!\left[\max_x \varphi\right]\!\right]_{(\kappa,\upsilon)} t = \begin{cases} \max \mathfrak{m} & \text{if } \mathfrak{m} \neq \emptyset \\ 0 & \text{otherwise} \end{cases}.$$

Lemma 3. *If ϕ_1 is an RMTL-$\int_3$ formula then $[\![\phi_1]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_1]\!](\kappa,\upsilon,t)$.*

Proof. We will prove by mutual structural induction that $[\![\phi_1]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_1]\!](\kappa,\upsilon,t)$ for any RMTL-$\int_3$ formula ϕ_1, and $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![\eta_1]\!](\kappa,\upsilon)\,t$ for any term η_1. For terms we have to prove that the following cases hold.

1. (Base Case α) If $\mathscr{T}[\![\alpha]\!]_{3(\kappa,\upsilon)\,t} = \mathscr{T}[\![\alpha]\!](\kappa,\upsilon)\,t$ then $\mathscr{T}[\![\alpha]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![\alpha]\!](\kappa,\upsilon)\,t$
2. (Base Case x) If $\mathscr{T}[\![x]\!]_{3(\kappa,\upsilon)\,t} = \mathscr{T}[\![x]\!](\kappa,\upsilon)\,t$ then $\mathscr{T}[\![x]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![x]\!](\kappa,\upsilon)\,t$
3. (Step Case $\int^{\eta_1} \phi_1$) If $[\![\phi_1]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_1]\!](\kappa,\upsilon,t)$, $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![\eta_1]\!](\kappa,\upsilon)\,t$, and $\mathscr{T}[\![\eta_1]\!](\kappa,\upsilon)\,t < 0$ iff $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} = \bot_{\mathbb{R}} \vee \mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} < 0$ then $\mathscr{T}\left[\!\left[\int^{\eta_1} \phi_1\right]\!\right]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}\left[\!\left[\int^{\eta_1} \phi_1\right]\!\right](\kappa,\upsilon)\,t$
4. (Step Case $\eta_1 \circ \eta_2$) If $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![\eta_1]\!](\kappa,\upsilon)\,t$ and $\mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![\eta_2]\!](\kappa,\upsilon)\,t$ then $\mathscr{T}[\![\eta_1 \circ \eta_2]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![\eta_1 \circ \eta_2]\!](\kappa,\upsilon)\,t$

Base cases 1 and 2 are trivially solved since by definition the semantic rules are exactly the same, and then for any model $\mathscr{T}[\![\alpha]\!]_{3(\kappa,\upsilon)\,t} = \mathscr{T}[\![\alpha]\!](\kappa,\upsilon)\,t$ and $\mathscr{T}[\![x]\!]_{3(\kappa,\upsilon)\,t} = \mathscr{T}[\![x]\!](\kappa,\upsilon)\,t$ hold. Step case $\int$. Assuming that $[\![\phi_1]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_1]\!](\kappa,\upsilon,t)$ and that $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![\eta_1]\!](\kappa,\upsilon)\,t$, we need to consider when the evaluation of term η_1 is less than zero. From the semantic nature of the term $\int^{\eta_1} \phi_1$ we have that for any model $\mathscr{T}\left[\!\left[\int^{\eta_1} \phi_1\right]\!\right]_{3(\kappa,\upsilon)\,t} = \bot_{\mathbb{R}}$ if $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} = \bot_{\mathbb{R}}$ and that $\mathscr{T}\left[\!\left[\int^{\eta_1} \phi_1\right]\!\right](\kappa,\upsilon)\,t = 0$ if $\mathscr{T}[\![\eta_1]\!](\kappa,\upsilon)\,t < 0$. Then from $\mathscr{T}\left[\!\left[\int^{\eta_1} \phi_1\right]\!\right](\kappa,\upsilon)\,t = 0$ iff $\mathscr{T}\left[\!\left[\int^{\eta_1} \phi_1\right]\!\right]_{3(\kappa,\upsilon)\,t} = \bot_{\mathbb{R}} \vee \mathscr{T}\left[\!\left[\int^{\eta_1} \phi_1\right]\!\right]_{3(\kappa,\upsilon)\,t} = 0$, we concude that $\mathscr{T}[\![\eta_1]\!](\kappa,\upsilon)\,t < 0$ iff $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} = \bot_{\mathbb{R}} \vee \mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} < 0$ holds for any model $(\kappa,\upsilon)t$. The step case 4 is direct.

Now, we continue the proof for formulas. We need to consider the cases:

1. (Base Case *true*) If $[\![true]\!]_{3(\kappa,\upsilon,t)} = [\![true]\!](\kappa,\upsilon,t)$ then $[\![true]\!]_{3(\kappa,\upsilon,t)} \prec [\![true]\!](\kappa,\upsilon,t)$
2. (Base Case p) If $[\![p]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$ iff $[\![p]\!](\kappa,\upsilon,t) = \mathsf{tt}$ then $[\![p]\!]_{3(\kappa,\upsilon,t)} \prec [\![p]\!](\kappa,\upsilon,t)$
3. (Step Case $<$) If $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![\eta_1]\!](\kappa,\upsilon)\,t$, $\mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![\eta_2]\!](\kappa,\upsilon)\,t$, $\mathscr{T}[\![\eta_1]\!](\kappa,\upsilon)\,t = \mathsf{ff}$ iff $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} = \bot_{\mathbb{R}} \vee \mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} = \mathsf{ff}$, and $\mathscr{T}[\![\eta_2]\!](\kappa,\upsilon)\,t = \mathsf{ff}$ iff $\mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} = \bot_{\mathbb{R}} \vee \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} = \mathsf{ff}$ then $[\![\eta_1 < \eta_2]\!]_{3(\kappa,\upsilon,t)} \prec [\![\eta_1 < \eta_2]\!](\kappa,\upsilon,t)$
4. (Step Case $\neg$) If $[\![\phi_1]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_1]\!](\kappa,\upsilon,t)$ and $[\![\phi_1]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$ iff $[\![\neg\phi_1]\!]_{3(\kappa,\upsilon,t)} = \mathsf{ff}$ and $[\![\phi_1]\!](\kappa,\upsilon,t) = \mathsf{tt}$ iff $[\![\neg\phi_1]\!](\kappa,\upsilon,t) = \mathsf{ff}$ then $[\![\neg\phi_1]\!]_{3(\kappa,\upsilon,t)} \prec [\![\neg\phi_1]\!](\kappa,\upsilon,t)$
5. (Step Case $\vee$) If $[\![\phi_1]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_1]\!](\kappa,\upsilon,t)$, $[\![\phi_2]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_2]\!](\kappa,\upsilon,t)$, $[\![\phi_1]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt} \vee [\![\phi_2]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$ iff $[\![\phi_1 \vee \phi_2]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$ and $[\![\phi_1]\!](\kappa,\upsilon,t) = \mathsf{tt} \vee [\![\phi_2]\!](\kappa,\upsilon,t) = \mathsf{tt}$ iff $[\![\phi_1 \vee \phi_2]\!](\kappa,\upsilon,t) = \mathsf{tt}$ then $[\![\phi_1 \vee \phi_2]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_1 \vee \phi_2]\!](\kappa,\upsilon,t)$

6. (Step Case $\mathrm{U}_{\sim\gamma}$) If $[\![\phi_1]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_1]\!]_{(\kappa,\upsilon,t)}$, $[\![\phi_2]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_2]\!]_{(\kappa,\upsilon,t)}$, and $[\![\phi_1 \,\mathrm{U}_{\sim\gamma}\, \phi_2]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$ iff $[\![\phi_1 \,\mathrm{U}_{\sim\gamma}\, \phi_2]\!]_{(\kappa,\upsilon,t)} = \mathsf{tt}$, then $[\![\phi_1 \,\mathrm{U}_{\sim\gamma}\, \phi_2]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_1 \,\mathrm{U}_{\sim\gamma}\, \phi_2]\!]_{(\kappa,\upsilon,t)}$

7. (Step Case $\exists\, x$) If there exists a model (κ, υ, t) such that $[\![\phi_1]\!]_{3(\kappa,\upsilon,t)} \prec [\![\phi_1]\!]_{(\kappa,\upsilon,t)}$, $[\![\exists\, x\, \phi_1]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$ iff $[\![\exists\, x\, \phi_1]\!]_{(\kappa,\upsilon,t)} = \mathsf{tt}$, then $[\![\exists\, x\, \phi_1]\!]_{3(\kappa,\upsilon,t)} \prec [\![\exists\, x\, \phi_1]\!]_{(\kappa,\upsilon,t)}$

We trivially prove that $[\![true]\!]_{3(\kappa,\upsilon,t)} = [\![true]\!]_{(\kappa,\upsilon,t)}$, since the semantic definition of *true* in both logics is the same. Base case p. From the semantic nature of p, we prove that $[\![p]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$ iff $[\![p]\!]_{(\kappa,\upsilon,t)} = \mathsf{tt}$ holds, since $[\![p]\!]_{3(\kappa,\upsilon,t)} = \mathsf{tt}$ iff $p \in \kappa(t)$ and $[\![p]\!]_{(\kappa,\upsilon,t)} = \mathsf{tt}$ iff $p \in \kappa(t)$. Step case $<$. Assuming $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![\eta_1]\!]_{(\kappa,\upsilon)\,t}$ and $\mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} \lhd \mathscr{T}[\![\eta_2]\!]_{(\kappa,\upsilon)\,t}$, we need to prove that $\mathscr{T}[\![\eta_1]\!]_{(\kappa,\upsilon)\,t} = \mathsf{ff}$ iff $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} = \bot_{\mathbb{R}} \vee \mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} = \mathsf{ff}$ and $\mathscr{T}[\![\eta_2]\!]_{(\kappa,\upsilon)\,t} = \mathsf{ff}$ iff $\mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} = \bot_{\mathbb{R}} \vee \mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} = \mathsf{ff}$ hold. For simplicity, we consider the proposition $\mathscr{T}[\![\eta_1]\!]_{3(\kappa,\upsilon)\,t} = \mathsf{tt}$ iff $\mathscr{T}[\![\eta_1]\!]_{(\kappa,\upsilon)\,t} = \mathsf{tt}$ and $\mathscr{T}[\![\eta_2]\!]_{3(\kappa,\upsilon)\,t} = \mathsf{tt}$ iff $\mathscr{T}[\![\eta_2]\!]_{(\kappa,\upsilon)\,t} = \mathsf{tt}$. For these cases the semantic rules are the same, and then the proposition holds. Proofs for step cases $\neg$, $\vee$, U, and $\exists$ are skipped since they are direct. □

Before concluding this section, we define a function to translate formulas containing minimum and maximum terms into formulas without occurrences of these operators.

Definition 9 (erasure of min/max terms)**.** Let f_ϕ and f_η be two mutually recursive functions responsible for erasing minimum and maximum terms from formulas and terms, respectively. In the case when ϕ_1 (the recursive argument of f_ϕ) is of the form

$$\eta_3 + \eta_4 \times \min_x \phi_1 < \eta_5$$

then the function returns

$$(\forall y\ y < x \rightarrow \neg f_\phi(\phi_1[y/x])) \wedge f_\phi(f_\eta(\eta_3 + \eta_4 \times x) < f_\eta(\eta_5)).$$

Otherwise, the function f_ϕ proceeds recursively over its sub-formulas and f_η over its sub-terms until no more occurrences of min/max terms exists.

Note that due to verbosity in the above definition, the formula $\eta_3 + \eta_4 \times \min_x \phi_1 < \eta_5$ does not extent with the $\neg$ and $\vee$ operators, since any inequality in a formula will reduce to this pattern using the connectives properties. For terms, any term will reduce to the pattern $\eta_3 + \eta_4 \times x$ using commutative and distributive properties of addition and multiplication.

Lemma 4. *The function f_t is partially correct.*

Proof Sketch. The proof follows by mutual structural induction on the formulas and terms containing min/max terms, and using axioms A1 and A2. □

3.3 Polynomial Inequality Translation

A close examination of the semantics of RMTL-$\int_3$ reveals that the timed state sequence κ and the logic environment υ are not directly related as parameters for evaluating the truth value of formulas. This property allows us to define a mechanism for introducing isolation by splitting formulas and/or translating them into polynomial inequality conditions. Several conditions can be discarded prior to execution, and the resulting simplified formula is then suitable for runtime monitoring and/or checking with SMT solvers.

The axiom system for the arithmetic of real numbers provided by Tarski [Tarski, 1995] can be used to encode polynomial inequalities as in RMTL-$\int_3$. Several properties provided by this well-known fragment will be used to facilitate the removal of quantifiers, when properties expressed as quantified formulas are monitored at execution time. From the Tarski–Seidenberg theorem [Tarski, 1995] we have that for any formula in $\text{FOL}_{\mathbb{R}}$, there exists an equivalent one not containing any existential quantifiers. Thus it is possible to define a decision procedure for quantifier elimination over $\text{FOL}_{\mathbb{R}}$. One of the most efficient algorithms, with complexity 2-EXPTIME, is *cylindrical algebraic decomposition* (CAD), later proposed by Collins [Collins, 1976, Basu et al., 2006]. To use it we require a set of axioms for isolation of temporal operators and duration terms, and an automatic mechanism to apply them.

Let us now describe the constraint required for an RMTL-$\int_3$ formula to be interpreted as a formula of $\text{FOL}_{\mathbb{R}}$; and the notion of rigid term and rigid formula.

Definition 10 (Inequality Translation Constraint)**.** Let ϕ_3 be a formula in RMTL-$\int_3$. ϕ_3 is a formula in $\text{FOL}_{\mathbb{R}}$ if it is free of duration terms, minimum/maximum terms, temporal operators, and propositions.

Definition 11 (Rigid Formula)**.** A term r is said to be rigid if its evaluation does not depend on the model parameter t. A *rigid formula* ϕ_r is a formula where every term is a rigid term.

In what follows, let $\phi_<$ be a formula containing a conjunction of polynomial inequalities of the form $T^1 < T^2 \wedge T^3 < T^4 \wedge \cdots \wedge T^{n-1} < T^n$ with T a term and $\frac{n}{2}$ the number of inequalities; $\phi_{\not<}$ a formula free of polynomial inequalities; and ϕ_i a formula of RMTL-$\int_3$ with index $i \in \mathbb{N}$.

Definition 12 (DNF_3 Formula)**.** A formula $\phi_i \in \Phi^3$ is in DNF_3 if the subformulas of the until operators and the duration terms are in DNF_3, or it is a formula not containing occurrences of until operators and duration terms, in *disjunctive normal form* (DNF).

Axioms A4 and A5 below describe how rigid formulas ϕ_r can be isolated outside the scope of the temporal operator. Axiom A6 isolates polynomial inequalities inside duration terms. Axiom A7 isolates inequalities inside duration terms.

A 4. $\phi_1 \vee (\phi_r \wedge \phi_2)\ \mathrm{U}_{\sim\gamma}\ \phi_3 \iff (\phi_r \to \phi_1 \vee \phi_2\ \mathrm{U}_{\sim\gamma}\ \phi_3) \wedge (\neg\phi_r \to \phi_1\ \mathrm{U}_{\sim\gamma}\ \phi_3)$

A 5. $\phi_1\ \mathrm{U}_{\sim\gamma}\ (\phi_r \wedge \phi_2) \vee \phi_3 \iff (\phi_r \to \phi_1\ \mathrm{U}_{\sim\gamma}\ \phi_2 \vee \phi_3) \wedge (\neg\phi_r \to \phi_1\ \mathrm{U}_{\sim\gamma}\ \phi_3)$

A 6. $\int^r \phi_r \wedge \phi \sim \eta \iff \left(\phi_r \wedge \int^r \phi \sim \eta\right) \vee (\neg\phi_r \wedge 0 \sim \eta)$

A 7. $\Box \int^\eta \phi_1 \vee \phi_2 = \int^\eta \phi_1 + \int^\eta \phi_2 - \int^\eta \phi_1 \wedge \phi_2$

Soundness proofs for axioms A4, A5, A6, A7 can be found in Appendix D. These axioms are used to provide isolation of formulas for certain patterns, but an automated method is required to apply them. Due to the changing nature of temporal operators and the duration terms over the model parameter t, this method is not straightforward and several details should be considered. First, we need to consider that duration terms inside until operators cannot be isolated but can be simplified. The nature of these operators does not allow for splitting a conjunction/disjunction of two different formulas as is thew case for rigid terms inside until operators. They can however be split using axiom A4 and/or A5. Terms occurring inside duration terms can be split by axiom A6, A3 and/or A7.

Definition 13 (Isolated Formula)**.** A formula ϕ_i is said to be *isolated* if every term and temporal operator depending on the parameter t does not contain other terms or temporal operators depending on the model parameter t.

Definition 14 (Simplified formula)**.** A formula is said to be *simplified* if the quantified polynomial inequalities have been decomposed and all variables are bounded. A simplified formula is a formula where operators and terms depending on the parameter model t only contain equalities of the form $x = \int^\eta \varphi$.

The resulting formula of our process shall be a simplified formula. Second, any formula produced by our automated method cannot contain logic variables that are free. The presence of free variables would mean that the monitor should solve a satisfiability problem on the fly, which is not admissible for our purpose. We should solve as many formulas as possible offline, and avoid formulas containing free variables (these are corner cases that will receive a different treatment). Lastly, we need to consider that temporal operators shall be mapped to propositions, and duration terms to free variables. Propositions shall be mapped to $x = 1$ for an arbitrary logic variable x.

We also prove, in Lemmas 5 and 6, that any formula of the form $\phi^1\ \mathrm{U}_{\sim\gamma}\ \phi^2$ or $\eta \sim \int^{\eta_x} \phi$ can be simplified. Proofs are also given in Appendix D. Some definitions and intermediate lemmas are included in Appendix D as well.

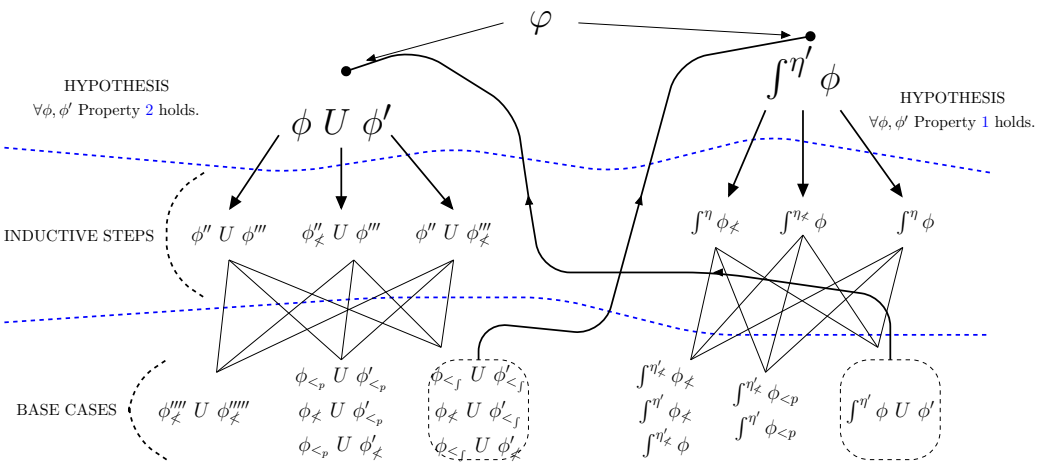


Figure 3.1: Graphical proof sketch

Lemma 5. *Let ϕ^1, ϕ^2 be two formulas in RMTL-$\int_3$ and consider the formula $\phi^1 \, \mathrm{U}_{\sim\gamma} \, \phi^2$. Then, there exists an equivalent formula where every until operator is free of inequalities or only contains equalities of the form $x = \int^{\eta} \varphi$.*

Lemma 6. *Let ϕ be a formula in RMTL-$\int_3$, and η_x, η two terms, and consider the formula $\eta \sim \int^{\eta_x} \phi$. Then, there exists an equivalent formula where any duration term is free of inequalities, or only contains equalities of the form $x = \int^{\eta} \phi$.*

Theorem 1. *Let ϕ be a RMTL-$\int_3$ formula. For any formula ϕ, there exists an equivalent simplified formula.*

Before presenting the proof of Theorem 1, let us give an intuitive proof sketch for it. The proof idea is to ensure that the existential quantifiers of a RMTL-$\int_3$ formula are removed, and the remaining inequalities are isolated to give us a simplified formula. Figure 3.1 shows the relations/dependences of Lemma 5 and Lemma 6 that are used in parts of the proof of Theorem 1. The figure shows that two main inductive hypotheses are applied for both branches, based on Property 1 and Property 2 that are introduced next. They refer to formulas and terms which are mutually recursive. Before introducing those properties, let us introduce some required definitions.

Definition 15. Let $f_\phi(X, Y, Z)$ be a shorthand for $(X \to Y) \wedge (\neg X \to Z)$, where X, Y and Z are formulas in RMTL-$\int_3$.

Let $\mathbf{f}^{\not<}$ be a map function from a formula ϕ in RMTL-$\int_3$ to a formula free of inequalities, or at most containing equalities of the form $x = \int^{\eta} \phi_2$, where ϕ_2 is a sub-formula of ϕ. Let $\mathbf{f}^{<}$ be a map function from a formula ϕ in RMTL-$\int_3$ to a formula $\phi_<$ with arbitrary length n. We denote by $\mathbf{f}_i^{\not<}$ and $\mathbf{f}_i^{<}$ map functions for arbitrary identifiers $i \in \mathbb{N}$. Note that

defining the translation in a sequence of small mappings will ease the proof structure of the Lemma 5. We also denote $\mathbf{f}_i^{\diamond}$ with $\diamond \in \{<, \not<\}$.

Definition 16. Let $S^{\diamond}_{(\phi^n_k)}$ be a set of formulas containing a combination of n disjunctions $\{\mathbf{f}_1^{\diamond}(\phi), \cdots, \mathbf{f}_n^{\diamond}(\phi)\}$ taken $k \leq n$ at a time without repetition, and $s^{\diamond}_{(\phi^n_k)}$ an element of the set $S^{\diamond}_{(\phi^n_k)}$.

Definition 17. Let $f^{\diamond}_{s_{(\phi^n_k)}} : \mathbb{N} \to S^{\diamond}_{(\phi^n_k)}$ a function such that $f^{\diamond}_{s_{(\phi^n_k)}}(i)$ is the i^{th} element of the set $S^{\diamond}_{(\phi^n_k)}$.

Definition 18. The intermediate function $f_{d_{(n)}} : \mathcal{V}^n \to \left(\mathbb{N} \to S^{\diamond}_{(\phi^n_k)}\right) \to \mathbb{N}^2 \to \Phi$ is defined by

$$f_{\phi}\left(f^{<}_{s_{(\phi^n_r)}}(1),\ y_{1,i} = \int^{\eta_x} f^{\not<}_{s_{(\phi^n_r)}}(1),\ y_{1,i} = 0\right) \wedge \ \cdots \ \wedge$$
$$f_{\phi}\left(f^{<}_{s_{(\phi^n_r)}}(m),\ y_{m,i} = \int^{\eta_x} f^{\not<}_{s_{(\phi^n_r)}}(m),\ y_{m,i} = 0\right),$$

where $y_{m,i} \in \mathcal{V}^n$, and $(m, i) \in \mathbb{N}^2$.

The following properties will allow us to simplify/transform terms and formulas by isolating inequalities from them. The isolation property for the sub-terms of the duration terms is presented as well.

Property 1 (Until Formula Isolation)**.**

$$\phi \,\mathrm{U}_{\sim\gamma}\, \psi \iff X_m$$

where X_i is defined as

$$\left(\mathbf{f}_i^{<}(\psi) \wedge \mathbf{f}_i^{<}(\phi) \wedge \left(\mathbf{f}_i^{\not<}(\phi)\, \mathrm{U}_{\sim\gamma}\, \mathbf{f}_i^{\not<}(\psi)\right)\right) \vee X_{i-1},$$

and $0 < i \leq m$, $m \in \mathbb{N}$.

Property 2 (Duration Term Isolation)**.**

$$\eta \sim \int^{\eta_x} \phi \iff Y_n \sim \eta \ \wedge D_n,$$

where Y_i is inductively defined by

$$(y_{1,i} + \cdots + y_{m,i}) - (Y_{i-1}),$$

D_i is inductively defined by

$$f_{d_{(n)}}\left((y_{1,i}, \ldots, y_{m,i},),\ f^{\diamond}_{s_{(\phi^n_r)}},\ (m, i)\right) \wedge D_{i-1},$$

$0 < i \leq n$, $m = \frac{n!}{r!\ (n-r)!}$, *and* $r = n - (i-1)$.

Proof Sketch of Theorem 1. The proof follows by mutual structural induction on the formula ϕ and the term η. The case when ϕ is $\phi_1 \, \mathrm{U}_{\sim\gamma} \, \phi_2$ or η is $\int^{\eta_1} \phi_1$ is directly proved by applying Lemmas 5 and 6, respectively. For the reamining cases $true, p, \neg, \vee, \exists$ and for term cases $\alpha, x, \circ$, we have to prove that no relation exists between these rules and the model parameter t, i.e, the parameter t is always constant with respect to the evaluation of these formulas and terms.

The proofs for base formulas $true$ and p are trivial since t is fixed by the semantic rule. Let us now consider the case when ϕ is $\neg\phi_1$. From the semantic interpretation of RMTL-$\int_3$, we have that $[\![\neg\phi_1]\!]_{3(\kappa,\upsilon,t)}$ and $[\![\phi_1]\!]_{3(\kappa,\upsilon,t)}$ are evaluated at the same time instant t. In the case when ϕ is $\phi_1 \vee \phi_2$, we also have ϕ_1 and ϕ_2 evaluating at the same time t.

Finally, for the case when ϕ is $\exists x \, \phi_1$ we have to prove that if the formula ϕ_1 does not contain operators and terms depending on the parameter model t or only contain equalities of the form $x = \int^{\eta} \varphi$ then from CAD we have a simplified formula. This comes from straightfoward induction on ϕ_1 and from the assumption that CAD is sound. □

3.3.1 Simplification Algorithm

Based on Theorem 1, we know that there exists a decision procedure for simplifying formulas. To translate any formula in RMTL-$\int_3$ into a formula in $\mathrm{FOL}_{\mathbb{R}}$ compliant with Definition 10, we require an algorithm for generating simplified monitoring conditions. Algorithm 1 can be used to replace duration terms by new free variables constrained by the nature of those terms, with propositions being replaced by fixed-valued logic variables (e.g., $p = 1$ means that the proposition P is required for evaluation in a certain formula). The algorithm begins by testing if a formula contains free logic variables and existential quantifiers. If the formula can be simplified we proceed, otherwise we return the input formula ϕ_1 (Line 3). Next, the duration terms are recursively replaced by new fresh variables in υ, minimum and maximum terms are transformed into quantified inequalities, and inequality conditions are generated (Line 5). The function `reduce_fm` applies min/max term substitutions as provided by axioms A1, A2, and A3; `replace_fm` and `replace_tm` are functions that replace temporal operators and duration terms with new free variables and propositions (Line 4) and construct a set of subformulas and subterms to be mapped; and the auxiliary mutually recursive functions `map` and `solve` translate formulas in RMTL-$\int_3$ into $\mathrm{FOL}_{\mathbb{R}}$ formulas ready to be decomposed using *cylindrical algebraic decomposition* (CAD) (Line 6). The function `map` generates the polynomial inequality conditions for temporal operators and duration terms using axioms A4, A5, A6, and A7. Before submitting the resulting conditions to decomposition, all propositions are replaced by equalities of the form $p = 1$. Let us now see four example applications of the

Require: a formula ϕ_1
Ensure : a simplified formula ϕ_2

1 **Function** `simplify` *(ϕ_1)* **is**
begin
2 **let** $\phi_3 = \texttt{reduce\_fm}(\phi_1)$ **in**
3 **if** $\texttt{is\_var\_free}(\phi_3)$ **then** ϕ_3 **else**
4 **let** $u\_set = \texttt{replace\_fm}(\phi_3)$ **in**
5 **let** $s\_set = \texttt{map}(u\_set, \emptyset)$ **in**
6 **let** $\phi_4 = \texttt{CAD}(\texttt{select}(s\_set))$ **in**
7 $\texttt{reduce}((s\_set \backslash \{\texttt{select}(s\_set)\}) \cup \{\phi_4\})$
end
8 **Function** `map` *(u_set,s_set)* **is**
begin
9 **if** $u\_set = \emptyset$ **then** s_set **else**
10 **let** $x = \texttt{select}(u\_set)$ **in**
11 **case** x **of**
begin
12 $x = \int^{\eta_1} \phi_6$ **:**
13 solve(sD(η_1,ϕ_6), u_set, s_set)
14 $\phi_7 \, \mathrm{U}_v \, \phi_8$ **:**
15 solve(sU(v,ϕ_7,ϕ_8), u_set, s_set)
16 ϕ_9 **:**
17 solve(sF(ϕ_9), u_set, s_set)
end
end
18 **Function** `solve`*(S,u_set,s_set)* **is**
begin
19 **if** (**let** (y,v) = S **in** v) **then**
20 **let** $u\_n = u\_set \backslash \{x\}$ **in**
21 $\texttt{map}(u\_n, s\_set \cup y)$
else
22 **let** $u\_n = u\_set \backslash \{x\}$ **in**
23 $\texttt{map}(u\_n \cup y, s\_set)$
end
24 **Function** sU*(a,ϕ_1,ϕ_2)* **is**
begin
25 **let** $(\texttt{ln}, \texttt{lw}) = \texttt{isol\_disj}(\texttt{dnf\_fm}(\phi_1))$ **in**
26 **if** $\texttt{lw} \neq []$ **then**
27 apply_axiom(a4_prim, a,
lst_to_dnf(ln), lw, ϕ_2)
else
28 **let** $(\texttt{ln2}, \texttt{lw2}) = \texttt{isol\_disj}(\texttt{dnf\_fm}(\phi_2))$ **in**
29 **if** $\texttt{lw2} \neq []$ **then**
30 apply_axiom(a5_prim, a,
lst_to_dnf(ln2), lw2, ϕ_1)
31 **else**
$(\phi_1 \, \mathrm{U}_{<a} \, \phi_2, \ true)$
end
32 **Function** sD*(η_1, ϕ_1)* **is**
begin
33 **let** $(\texttt{ln}, \texttt{lw}) = \texttt{isol\_disj}(\texttt{dnf\_fm}(\phi_1))$ **in**
if $\texttt{len}(\texttt{lw}) > 1$ **then**
34 apply_axiom(a7_prim, η_1,
lst_to_dnf(ln), lw, ϕ_1)
else
35 **let** $(\texttt{ln}, \texttt{lw}) = \texttt{isol\_cnj}(\texttt{dnf\_fm}(\phi_1))$ **in**
if $\texttt{lw} \neq []$ **then**
36 apply_axiom(a6_prim, η_1,
lst_to_dnf(ln), lw, ϕ_1)
else
$(\int^{\eta_1} \phi_1, \ true)$
end
37 **Function** sF*(ϕ_1)* **is**
begin
38 **if** $\texttt{isIsolated}(\phi_1)$ **then** $(\phi_1, true)$ **else**
$(\phi_1, false)$
end

Algorithm 1: Simplification of RMTL-$\int_3$ Inequalities

algorithm.

Example 5. *Consider the duration formula*

$$0 < \int^{10} a \vee \phi_{<}.$$

The result of applying the function `replace_fm` *to this formula is the set containing the formulas* $0 < x$ *and* $x = \int^{10} a \vee \phi_{<}$. *Applying axiom A7 over the second formula results*

in

$$x + \int^{10} a \wedge \phi_< = \int^{10} a + \int^{10} \phi_<.$$

Getting decomposed the or operator inner the duration term, we are able to generate the inequality conditions using the axiom A6. They are

$$\phi_< \rightarrow x = \int^{10} a + \int^{10} true - \int^{10} (a \wedge true)$$

that simplifies to

$$\phi_< \rightarrow x = \int^{10} true$$

and

$$\neg\phi_< \rightarrow x = \int^{10} a.$$

Finally, the output formula is

$$0 < x \wedge \left(\phi_< \rightarrow x = \int^{10} true\right) \wedge \left(\neg\phi_< \rightarrow x = \int^{10} a\right).$$

Note that when we have a temporal operator a similar generation of the inequality conditions is performed, but this time using axioms A4 and A5.

Example 6. *Let us now see an example using a formula containing a temporal operator. Consider the formula*

$$x > 0 \wedge a \,\mathrm{U}_{<10}\, (b \wedge x < 10).$$

We first note that $a \,\mathrm{U}_{<10}\, (b \wedge x < 10)$ *can be converted to an equivalent formula of the form*

$$((x < 10) \rightarrow a \,\mathrm{U}_{<10}\, b) \;\wedge\; \neg(x < 10) \rightarrow a \,\mathrm{U}_{<10}\, \mathsf{ff}.$$

This result comes from the application of axiom A5. In DNF_3*, we have*

$$(x > 0 \wedge x < 10 \wedge a \,\mathrm{U}_{<10}\, b) \vee x > 0 \wedge \neg(x < 10) \wedge a \,\mathrm{U}_{<10}\, \mathsf{ff},$$

which simplifies to $0 < x < 10 \wedge a \,\mathrm{U}_{<10}\, b$.

After this step we have the inequality conditions ready to be simplified using the CAD technique (Line 6). The decomposed formula can then be reduced, or else the terms initially found in the original formula can be replaced back (Line 7).

Example 7. *Let us now see a complete application of the algorithm for a simple formula. Consider the formula*

$$x < \int^{x+1} (a \wedge x < 10),$$

with a *a proposition whose truth value depends on the model parameter* t*. Since the logic variable* x *is used both at the level of the relation operator of the formula and in the*

duration term, finding a valuation of x that satisfies the formula is not trivial; we can use our algorithm to generate inequality conditions, and reduce the latter conditions into an RMTL-$\int_3$ formula. We begin by replacing the term $\int^{x+1}(a \wedge x < 10)$ by y and apply axiom A3 on the same term. We get the formula

$$x < y \wedge w = x + 1 \wedge y = \int^{w} (a \wedge x < 10)\,.$$

Applying axiom A6 on the duration term, we have

$$\left(x < 10 \rightarrow y = \int^{w} a\right) \wedge (\neg(x < 10) \rightarrow y = 0)\,.$$

Replacing $y = \int^{w} a$ with the constraint $0 \leq y < w$, we have the final formula, ready for simplification,

$$x < y \wedge w = x + 1 \wedge (x < 10 \rightarrow 0 \leq y < w) \wedge (\neg(x < 10) \rightarrow y = 0)\,.$$

After simplification of the formula using CAD we get

$$\textit{true if } x \in]-1, 0[;\ \textit{and } x < \int^{1+x} a \textit{ if } x \in [0, 10[.$$

After applying the function `reduce`*, the free logic variables are recursively substituted following the structure of the formula, with the exception of x that remains unchanged. In the case that x is substituted by a duration term, then we have a decision procedure to compute the truth value of the term based on the outcome of the procedure; if x has not been replaced by a duration term and x is not quantified, then we need to universally or existentially quantify it explicitly, otherwise the formula cannot be synthesized into a monitor.*

The functions `sU`, `sD`, `sF` are responsible for applying axioms A4-A7, and will play a major role in the proof of correctness of the algorithm. `isol_disj`, `isol_cnj`, `isIsolated` and `dnf_fm` will be described later in this thesis.

Example 8. *Let us now see a final example, but now with emphasis on duration of durations. Consider the quantified formula*

$$\exists y \int^{\int^{10} \phi_1 + y + 1} \phi_2 < y.$$

We can apply Axiom 3 since the scope of the duration term $\int^{10} \phi_1$ is immutable, and we get

$$\exists y\ z = \int^{10} \phi_1 + y + 1 \wedge \int^{z} \phi_2 < y.$$

Continuing the process as in the previous example, we have

$$\exists y\ z = h + y + 1 \wedge m < y \wedge 0 \leq h < 10 \wedge 0 \leq m < z$$

and after applying CAD we get

$$\int^{z} \phi_2 < 10 \wedge 1 + \int^{10} \phi_1 + \int^{z} \phi_2 < z < 11 + \int^{10} \phi_1.$$

A way to compute this formula is decomposing it by z, h, m order as follows:

$$(1 < z < 11 \wedge 0 \leq \int^{10} \phi_1 < -1 + z \wedge 0 \leq \int^{z} \phi_2 < -1 - \int^{10} \phi_1 + z) \ \vee$$
$$(11 \leq z < 21 \wedge -11 + z < \int^{10} \phi_1 < 10 \wedge 0 \leq \int^{z} \phi_2 < -1 - \int^{10} \phi_1 + z).$$

Note the that this example cannot be submitted for monitoring purposes until the formula has no free variables and quantifiers. However, for solving it using an SMT solver it is possible as we will see in the next section.

3.3.2 Functional Correctness

To ensure that the above algorithm correctly does what it is supposed to do, we begin by stating the functional correctness criteria, lemmas and theorems. Every lemma is guided by the required statements to conclude the proof of the functional correctness theorem. Some definitions and lemmas appear in Appendix D, due to their considerable length.

Lemma 7. *The function* `sU` *is partially correct.*

Proof. The proof follows by case analysis on the structure of function `sU`. We have three cases. The first one is when ϕ_1 contains inequalities. We have to prove that if `lw` is not empty then the application of the Axiom 4 is sound. The result came from the soundness of the Axiom 4 as the function `apply_axiom` (Line 27) applies explicitly the axiom. The second case is when ϕ_1 is free of inequalities, and ϕ_2 contains inequalities. We have to proof that if `lw2` is not empty then the application of the Axiom 5 is sound. The proof cames from the soundness of this axiom as stated in Appendix D. The third case is when ϕ_1 and ϕ_2 do not contain formulas with inequalities. We have to prove that if `lw` and `lw2` is empty then *true* is returned meaning that no changes have been performed in ϕ_1 neither in ϕ_2. The proof is trivial. We conclude the proof that for a given input set there is an output formula which is equal to the input formula, or totally/partially simplified. □

Lemma 8. *The function* `sD` *is partially correct.*

Proof. The proof is similar to the proof of the Lemma 7. □

Lemma 9. *The function* `map` *is partially correct.*

Proof. The proof follows by case analysis on S (Line 19).

The function `map` takes as input a set *u_set* of formulas and a set *s_set* of simplified formulas, and calls one of the functions `sU`, `sD`, or `sF`, as appropriate, to process one of the formulas of *u_set*. Recall that the atomic simplification functions `sD/sU/sF` may need to be applied more than once to a given formula; for this reason the functions return a pair consisting of a simplified formula and a boolean indicating whether the formula has been fully simplified (in which case no further calls are required). Depending on whether the selected formula has been fully simplified or not, it will be moved (or not) to the set *s_set* of simplified formulas. The auxiliary function `solve` takes a formula returned by `sD/sU/sF` and recursively calls `map` modifying *u_set* and *s_set* as appropriate.

- Case S always return v equals true:

  As the unsolved set (*u_set*) decreases and the solved set (s_set) increases until *u_set* is empty, we have that all formulas are solved. The functional correctness depends then on the partially correctness of the functions `sU`, `sD`, and `sF` given by Lemmas 7 and 8, respectively.

- Case S does not always returns v equals false:

  From the assumption that the function S is partially correct, we have that there is no other path for terminating the recursive calls than at some point in the execution of the function `solve`, the function S returns a solved formula several enough times to solve all the subformulas. From that, we have to prove that if the function `map` returns then the solved set has increased with correct solved formulas and the unsolved set has decreased in the same ratio. Then, the correctness of the resulting formula depends on the partially correctness of the functions `sU` and `sD` that is given by Lemmas 7 and 8, and also on the correctness of the function `sF`. The partially correctness of this function is straightforward since it only returns a solved formula if the formula contains every subformula in the solved set. Finally, we have that "if the function `map` returns then it returns a tuple containing a formula processed by applying sound axioms and a true value" holds.

Hence, the correctness proof ends since the `map` function holds both cases. □

Let us now introduce the theorem to state that the Algorithm 1 simplify RMTL-$\int_3$ formulas as expected, i.e., for each input the algorithm produces the expected output.

Theorem 2 (Functional Correctness)**.** *For all input formulas of the Algorithm 1, if the Algorithm 1 returns a formula then this formula is simplified.*

Proof of Theorem 2. Let us denote the pre condition p meaning the algorithm returns, and the post condition q meaning that the output is a simplified formula. We have to prove that p implies q. We proceed by directly prove that the sequential statements of the `simplify` function are partially correct. We begin by proving that the function `reduce_fm` is partially correct, which result came from Lemma 4. Case when `is_var_free` return true then the function returns the formula ϕ_1 without minimum and maximum terms. Otherwise, we have to prove that if the function `replace_fm` returns then the output is a tuple containing two sets of formulas *u_set* and *s_set*. We skip this proof step. Next, we prove that `map` is partially correct as stated by Lemma 9. We skip the proof step for Colin's CAD since it is well know and established algorithm. We also omit the proof step for `reduce` since it makes the reverse of the function `replace_fm`. Hence, Algorithm 1 returns simplified formulas. □

Theorem 3 (Termination)**.** *For all input formulas, the Algorithm 1 terminates.*

Proof of Theorem 3. We only consider the termination proof step for the function `map`, and skip the remaining direct proof steps. As the proof for the Lemma 9, this proof has the same shape for the case analysis.

- Case S always return v equals true:

  As the unsolved set decreases (u_set) and the solved set (s_set) increases until it is empty, we have that the `map` function is primitive recursive if S is also a primitive recursive function.

- Case S does not always returns v equals false: From that, we have to prove that if the function `map` returns then the solved set is eventually increasing with solved formulas and the unsolved set is decreasing. We also have to prove that successive calls of `sU`, `sD` and `sF` are upper bounded by the number of the inequalities in a formula and that these functions terminate.

  Let us now consider three inductive steps, one for each function application, and skip the base cases since they are trivial. From Lemma 11, successive calls of `sU` are upper bounded by $2^n - 1$, where n is the number of inequalities. Since n is finite, we have to apply those axioms finitely. For successive call of `sD`, we follow from Lemma 12 that give us also an upper bound. Finally, function `sF` only returns a formula if every sub-formula is solved. We have to prove that if no more successive calls of `sD` and `sU` can happen then the input formula of `sF` is a solved formula. This is a result stated in Theorem 1 that indirectly states that for any formula in RMTL-$\int_3$ there is an equivalent simplified formula by successive application of the axioms A3, A4, A5, A6, A7, which is chosen as the required pattern. Given the

shape of these axioms, we also have that the application order of the axioms do not impact the final formula and then no backtracking algorithm is required.

We conclude the proof with the statement that the function `map` terminates. Assuming that `CAD` ,`reduce_fm`, `replace_fm` and `reduce` terminate then Algorithm 3 terminates. □

To conclude, we guarantee that if the algorithm terminates then we have a simplified formula, and at same time that the algorithm is bounded and thus terminates for any formula, assuming that CAD terminates.

3.4 SMT Synthesis for RMTL-$\int_3$ Formulae

The synthesis algorithm for RMTL-$\int_3$ presented here is suitable for solve the satisfiability problem of our fragment using dyadic rationals (real numbers of the form $\frac{m}{2^n}$ for $n, m \in \mathbb{Z}$). This means that our formalization is adjusted as an input model for SMT solvers in SMT-LIBv2 specification language. At this point formulas shall be in simplified form. In the next section we will present an alternative algorithm that generates executable monitors.

SMT provers have been progressively adding smart tactics for solving problems that until now could only be solved using human creativity. Of course several issues such as inductive proofs and quantified fragments are really difficult or even impossible to check by such general approaches.

Due to being the target of several optimizations, such as conflit-driven clause learning, and also due to their efficiency handling a mix of non-quantified logic fragments, including non-interpreted functions and decidable logic fragments for arithmetic, these solvers are suited for several classic problems in the real-time community. This fact has not been suitably explored until now; we give here just steps in this direction.

Efficient synthesis algorithms can give modular advantages for different problem formulations such as schedulability analysis. In order to give a feasible *time model* for synthesis of RMTL-$\int_3$, we have to assume that intervals have exactly size one and symbols can be consecutively repeated in the input timed sequence, in order to formulate the new synthesis algorithm. This is a restriction over the time model used in interval-based semantics. We take this choice to avoid a more complex problem formulation and utilization of the solver's features that may induce the problem to be unfeasible at the first place due to make use of a more detailed timed model. We will now describe a new algorithm for synthesis of RMTL-$\int$ with this restricted model over interval-based semantics using lambda expressions, that will be converted to the SMT-LIBv2 [Barrett et al., 2010] language with small effort.

The set of theories that we use are *quantified uninterpreted functions with equality*, *arrays*, and *non-quantified non-linear arithmetic*. For arrays we use the *select* word that given a trace and a time t returns a proposition. $first$ and *second* constructs are used for pairs, and **ite** is the if-then-else construct. In what follows we define the combinators **evalP**, **evalU**, **evalD**, that will evaluate respectively propositions, less-until operator, and duration terms, based on the standard rewriting semantics of λ-expressions (β-reduction). The other operators available in RMTL-$\int_3$ are directly converted. These include the common $\neg$ and $\vee$ operators and the arithmetic operators $+$ and $\times$. The proposition formulation is encoded by the lambda expression

$$\textbf{evalP} \doteq \ \lambda\ p\ t\ .\ \textbf{ite}\ (\textbf{select}\ \kappa\ t = p)\ \mathtt{tt}\ \mathtt{ff},$$

where *select* word selects a given element of the array κ for some index and returns a proposition. κ is not propagated along the definitions in order to avoid being verbose. We encode the trace as an array and the time t as an index, meaning that time is discrete. The word **eval** should be replaced by one of the evaluation functions as appropriate. Evaluation of the less until is defined by the following set of lambda expressions

$$\begin{aligned}
map4 \quad &\doteq \lambda\ b\ .\ \textbf{ite}\ (b = \mathtt{tt})\ \mathtt{tt}\ (\textbf{ite}\ (b = \mathtt{ff})\ \mathtt{ff}\ \bot),\\
evali \quad &\doteq \lambda\ b1\ b2\ .\ \textbf{ite}\ (b2 \neq \mathtt{ff})\ (map4\ b2)\ (\textbf{ite}\ (b1 \neq \mathtt{tt})\ (map4\ b1)\ \mathfrak{r})),\\
evalb \quad &\doteq \lambda\ t\ v\ .\ \textbf{ite}\ (v = \mathfrak{r})\ (evali\ (\textbf{eval}\ t)\ (\textbf{eval}\ t))\ v,\\
evalf' \quad &\doteq \lambda f.\ \lambda x\ i\ .\ (x \geq 0) \rightarrow \textbf{ite}\ (i \geq 0 \wedge x > i)\\
&\qquad (evalb\ x\ ((f\ f)\ (x-1)\ i) = (f\ f)\ x\ i)\\
&\qquad (evalb\ x\ \mathfrak{r} = (f\ f)\ x\ i),\\
evalf \quad &\doteq evalf'\ evalf',\\
map3 \quad &\doteq \lambda x\ .\ \textbf{ite}\ ((first\ x = true) \wedge (second\ x = \mathfrak{r}))\ \bot\\
&\qquad (\textbf{ite}\ ((first\ x = false) \wedge (second\ x = \mathfrak{r}))\ \mathtt{ff}\ (\textbf{ite}\ (second\ x = \mathtt{ff}_4)\ \mathtt{ff}\ \mathtt{tt})),\\
evalc \quad &\doteq \lambda t\ t'\ .\ mkpair\ (trc_size \leq 10)\ (evalf\ (t-1)\ t'),\ and\\
\textbf{evalU} \quad &\doteq \lambda t'\ t\ .\ map3\ (evalc\ t\ t').
\end{aligned}$$

Evaluation of the duration term is defined by

$$\begin{aligned}
ind \quad &\doteq \lambda\ \kappa\ t\ .\ \textbf{ite}\ (\textbf{eval}\ t = \mathtt{tt})\ 1\ 0\\
evale' \quad &\doteq \lambda f\ .\ \lambda x\ i\ .\ (x \geq 0) \rightarrow \textbf{ite}\ ((i \geq 0) \wedge (x > i))\\
&\qquad (((f\ f)\ (x-1)\ i) + (ind\ \kappa\ x) = (f\ f)\ x\ i)\\
&\qquad (ind\ \kappa\ x = (f\ f)\ x\ i)\\
evale \quad &\doteq evale'\ evale'\\
\textbf{evalD} \quad &\doteq \lambda t'\ t\ .\ evale\ (t-1)\ t'
\end{aligned}$$

Note that we need to remove the recurrence among the lambda expressions by unfolding. To avoid us or the SMT solver unfolding so many times, a bound over quantification for the temporal and duration operators is applied, based on the temporal nature of the operator. For $\int^{\gamma_d} \phi$, we assume that the duration is in the interval $[t, t+\gamma_d[$ for all $t \in \mathbb{N}_0^+$, and for the case $\phi_1 \; U_{<\gamma} \; \phi_2$ we assume the interval $[t, t+\gamma[$ for all $t \in \mathbb{N}_0^+$. These assumptions help us to reduce the search space in order to generate at least one finite model. The following Example 9 illustrates this for a simple case.

Example 9. *The expression evale* 2 1 *will be evaluated as follows:*

$$\begin{aligned}
&evale\ 2\ 1 \longrightarrow_\beta \\
&(\lambda xi.(x \geq 0) \rightarrow ite(i \geq 0 \wedge x > i) \\
&((evale'\ evale')\ x\ i = ((evale'\ evale')(x-1)i) + (ind\ k\ x)) \\
&((evale' evale')\ x\ i = ind\ k\ x))\ 2\ 1 \longrightarrow_\beta^* \\
&(2 \geq 0 \rightarrow \textbf{ite}\ (1 \geq 0 \wedge 2 > 1) \\
&(evale'\ 2\ 1 = (1 \geq 0 \rightarrow \textbf{ite}\ (1 \geq 0 \wedge 1 > 1) \\
&(evale'\ 1\ 1 = (evale'\ 1\ 1 = ind\ \kappa\ 1) + (ind\ \kappa\ 1)) \\
&(evale'\ 1\ 1 = ind\ \kappa\ 1)) + (ind\ \kappa\ 2)) \\
&(evale'\ 2\ 1 = ind\ \kappa\ 2))
\end{aligned}$$

where after simplifying we get

$$evale'\ 2\ 1 \longrightarrow_\beta^* (evale'\ 1\ 1 = ind\ \kappa\ 1) + (ind\ \kappa\ 2).$$

One trick that can be used to encode such notations in SMT solvers logically consists of encoding such definitions by using uninterpreted functions and universal quantification. The uninterpreted function f_{evale} can be specified by writing the following axiom:

$$\begin{aligned}
\forall x\ i, \quad & (x \geq 0) \rightarrow \textbf{ite}\ ((i \geq 0) \wedge (x > i)) \\
& (f_{evale}\ x\ i = (f_{evale}\ (x-1)\ i) + (ind\ \kappa\ x)) \\
& (f_{evale}\ x\ i = ind\ \kappa\ x).
\end{aligned}$$

In this section we have presented a synthesis algorithm for the interval-based semantics of RMTL-$\int_3$ with a restricted model. We have adopted this restriction due to the simplicity and feasibility of the approach using array theory. Other alternatives may be used such as the codification of the interval-based semantics without such restrictions, but this may increase the burden for solving the same problem using a more refined timed model. As a last remark, we should note that the duration term can be bounded by all terms, not only for α and x. In what follows we will discuss a computable approach.

3.5 Computation of RMTL-$\int_3$ Formulae

This algorithm is able to generate monitors that can be directly executed on the target platform and draw a three-valued verdict, instead of deciding if there is a model that satisfies a given formula. Monitors are generated for functional programming languages but can be further converted to imperative languages such as C++11 with small effort, as we further describe in Appendix A. This algorithm encodes reals as floating point numbers.

Given the definition of RMTL-$\int_3$, we can derive an evaluation algorithm for monitor synthesis. In what follows we will present the algorithm and study the time complexity of the computation with respect to both trace and formula size.

We begin with a set of preliminary definitions. The set of timed sequences is denoted by $\mathbf{K}$, the duration of the timed state sequence $\kappa \in \mathbf{K}$ is denoted by $d^{(\kappa)}$, and the set of logic environments is denoted by Υ. Let $\mathbf{B}_4$ be the set $\{\mathtt{tt}_4, \mathtt{ff}_4, \bot_4\} \cup \{\mathfrak{r}\}$ where $\mathfrak{r}$ is a new symbol that will be used only for purposes of formulae evaluation, and $\mathbf{D}$ the set $\mathbb{R}_{\geq 0} \cup \{\bot_{\mathbb{R}}\}$. The function $sub_{\mathbf{K}} : (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \to \mathbb{R}_{\geq 0} \to \mathbf{K}$ defines a timed subsequence constrained by the interval $]t, t+\gamma]$, where t and γ are real numbers to be used as parameters in $sub_{\mathbf{K}}$. The function $map^{\mathbf{B}_4} : \mathbb{B}_3 \to \mathbf{B}_4$ maps $\mathtt{tt}$ to $\mathtt{tt}_4$, $\mathtt{ff}$ to $\mathtt{ff}_4$ and $\bot$ to $\bot_4$; $map^{\mathbb{B}_3} : \mathbb{B} \times \mathbf{B}_4 \to \mathbb{B}_3$ maps $(\mathtt{tt}, \mathfrak{r})$, $(\mathtt{tt}, \bot_4)$, and $(\mathtt{ff}, \bot_4)$ to $\bot$; $(\mathtt{ff}, \mathfrak{r})$, $(\mathtt{ff}, \mathtt{ff}_4)$, and $(\mathtt{tt}, \mathtt{ff}_4)$ to $\mathtt{ff}$; and $(\mathtt{ff}, \mathtt{tt}_4)$ and $(\mathtt{tt}, \mathtt{tt}_4)$ to $\mathtt{tt}$. We will employ a left $fold$ function defined in the usual way.

From a close examination of the operators, the corresponding Compute$_{(\neg)}$ and Compute$_{(\vee)}$ evaluation functions have time complexity constant in the number of timed sequence symbols, linear in the depth of the formula for Compute$_{(\neg)}$, and exponential in the depth of the formula for Compute$_{(\vee)}$. Let us consider the functions Compute$_{(\eta)}$ :: $(\mathbf{K} \times \Upsilon) \to \mathbb{R} \to \Gamma \to \mathbf{D}$ and Compute$_{(\varphi)}$ :: $(\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \to \Phi^3 \to \mathbb{B}_3$ for the evaluation of $U_<$ and $<$, and the term $\int$.

Operator $U_<$. Given formulas ϕ_1, ϕ_2 and $\gamma \in \mathbb{R}_{\geq 0}$, the formula $\phi_1 \, U_{<\gamma} \, \phi_2$ is evaluated in a model (κ, υ, t) by the function Compute$_{(U_<)}$: $(\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \to \mathbb{R}_{\geq 0} \to \Phi^3 \to \Phi^3 \to \mathbb{B}_3$, defined in Figure 3.2. We report here only on the computation function Compute$_{(U_<)}$; the remaining functions are Compute$_{(U_=)}$ for punctual until, Compute$_{(S_<)}$ for the non-punctual dual operator, and Compute$_{(S_=)}$ for the punctual dual operator. These operators have at most two new branches. Given an input κ with size n_κ, and a measure m_φ of the depth of a formula φ, we obtain from the structure of the computation the upper bound of time complexity $\binom{n_\kappa + m_\varphi}{m_\varphi} \cdot 2^{n_\kappa}$. For instance, we understand by a formula with depth one as $a \,\mathrm{U}\, b$, a formula with depth two as $(a \,\mathrm{U}\, b) \,\mathrm{U}\, (a \,\mathrm{U}\, b)$ and so on.

$$
\begin{array}{ll}
\mathrm{ev}^{\mathrm{i}}_{\mathrm{al}} & :: \mathbb{B}_3 \to \mathbb{B}_3 \to \mathbf{B}_4 \\
\mathrm{ev}^{\mathrm{i}}_{\mathrm{al}}\ b_1\ b_2 & \triangleq \begin{cases} map^{\mathbf{B}_4}\ b_2 & \text{if } b_2 \neq \mathsf{ff} \\ map^{\mathbf{B}_4}\ b_1 & \text{if } b_1 \neq \mathsf{tt} \text{ and } b_2 = \mathsf{ff} \\ \mathfrak{r} & \text{otherwise} \end{cases} \\
\mathrm{ev}^{\mathrm{b}}_{\mathrm{al}} & :: (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \to \Phi^3 \to \Phi^3 \to \mathbf{B}_4 \to \mathbf{B}_4 \\
\mathrm{ev}^{\mathrm{b}}_{\mathrm{al}}\ m\ \phi_1\ \phi_2\ v & \triangleq \begin{cases} \mathrm{ev}^{\mathrm{i}}_{\mathrm{al}} \left(\mathrm{Compute}_{(\varphi)}\ m\ \phi_1\right) \left(\mathrm{Compute}_{(\varphi)}\ m\ \phi_2\right) & \text{if } v = \mathfrak{r} \\ v & \text{otherwise} \end{cases} \\
\mathrm{ev}^{\mathrm{fold}}_{\mathrm{al}} & :: (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \to \Phi^3 \to \Phi^3 \to \mathbf{K} \to \mathbf{B}_4 \\
\mathrm{ev}^{\mathrm{fold}}_{\mathrm{al}}\ (\kappa, \upsilon, t)\ \phi_1\ \phi_2\ \varkappa & \triangleq fold\ \left(\lambda v\ (p, (i, t')) \to \mathrm{ev}^{\mathrm{b}}_{\mathrm{al}}\ (\kappa, \upsilon, t' - \epsilon)\ \phi_1\ \phi_2\ v\right) \mathfrak{r}\ \varkappa \\
ev^C_{al} & :: (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \to \mathbb{R}_{\geq 0} \to \Phi^3 \to \Phi^3 \to \mathbf{K} \to (\mathbb{B} \times \mathbf{B}_4) \\
ev^C_{al}\ (\kappa, \upsilon, t)\ \gamma\ \phi_1\ \phi_2\ \varkappa & \triangleq \left(d^{(\kappa)} \leq t + \gamma,\ \mathrm{ev}^{\mathrm{fold}}_{\mathrm{al}}\ (\kappa, \upsilon, t)\ \phi_1\ \phi_2\ \varkappa\right) \\
\mathbf{Compute}_{(U_<)}\ m\ \gamma\ \phi_1\ \phi_2 & \triangleq \begin{cases} map^{\mathbb{B}_3}\ \left(ev^C_{al}\ m\ \gamma\ \phi_1\ \phi_2\ \left(sub_{\mathbf{K}}\ m\ \gamma\right)\right) & \text{if } \gamma \geq 0 \\ \mathsf{ff} & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{ll}
ev^<_{al} & :: \mathbb{R} \to \mathbb{R} \to \mathbb{R} \\
ev^<_{al}\ val_1\ val_2 & \triangleq \begin{cases} val_1 < val_2 & \text{if } val_1 \in \mathbb{R} \text{ and } val_2 \in \mathbb{R} \\ \bot & \text{otherwise} \end{cases} \\
\mathbf{Compute}_{(<)}\ m\ h_1\ h_2 & \triangleq ev^<_{al} \left(\mathrm{Compute}_{(\eta)}\ m\ h_1\right) \left(\mathrm{Compute}_{(\eta)}\ m\ h_2\right)
\end{array}
$$

$$
\begin{array}{ll}
1_{\varphi(\kappa,\upsilon)} & :: (\mathbf{K} \times \Upsilon) \to \mathbb{R}_{\geq 0} \to \Phi^3 \to \{0, 1\} \\
1_{\varphi(\kappa,\upsilon)}\ (\kappa, \upsilon)\ t\ \phi & \triangleq \begin{cases} 1 & \text{if } \mathrm{Compute}_{(\varphi)}\ (\kappa, \upsilon, t)\ \phi = \mathsf{tt} \\ 0 & \text{otherwise} \end{cases} \\
ev^\eta_{al} & :: (\mathbf{K} \times \Upsilon) \to \Phi^3 \to \mathbf{K} \to \mathbb{R}_{\geq 0} \\
ev^\eta_{al}\ (\kappa, \upsilon)\ \phi\ \varkappa & \triangleq fold\left(\lambda s, (p, (i, t')) \to t' \cdot \left(1_{\varphi(\kappa,\upsilon)}\ (\kappa, \upsilon)\ t'\ \phi\right) + s\right) 0\ \varkappa \\
\mathbf{Compute}_{(\int)}\ (\kappa, \upsilon)\ t\ a\ \phi & \triangleq \begin{cases} ev^\eta_{al}\ (\kappa, \upsilon)\ \phi\ \left(sub_{\mathbf{K}}\ (\kappa, \upsilon, t)\ a\right) & \text{if } a \geq 0 \\ \bot_{\mathbb{R}} & \text{otherwise} \end{cases}
\end{array}
$$

Figure 3.2: Evaluation of the operators $U_<$ and $<$, and of duration terms

Function $\text{Compute}_{(\eta)}\ (\kappa, \upsilon)\ t\ h :: (\mathbf{K} \times \Upsilon) \to \mathbb{R} \to \Gamma \to \mathbf{D}$ **is**
  case h **of**

$$
\begin{array}{lcl}
\alpha & : & eval_\alpha\ \alpha \\
h_1 + h_2 & : & \left(\text{Compute}_{(\eta)}\ m\ h_1\right) + \left(\text{Compute}_{(\eta)}\ m\ h_2\right) \\
h_1 \times h_2 & : & \left(\text{Compute}_{(\eta)}\ m\ h_1\right) \times \left(\text{Compute}_{(\eta)}\ m\ h_2\right) \\
\int^{h_1} \phi & : & \text{Compute}_{(\int)}\ (\kappa, \upsilon)\ t\ \left(\text{Compute}_{(\eta)}\ (\kappa, \upsilon)\ t\ h_1\right)\ \phi
\end{array}
$$

  end
end

Function $\text{Compute}_{(\varphi)}\ m\ \phi :: (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \to \Phi^3 \to \mathbb{B}_3$ **is**
  case ϕ **of**

$$
\begin{array}{lcll}
p & : & eval_p\ m\ p & \text{– base case} \\
\neg\phi & : & \text{Compute}_{(\neg)}\ m\ \phi & \text{– Boolean operators} \\
\phi_1 \vee \phi_2 & : & \text{Compute}_{(\vee)}\ m\ \phi_1\ \phi_2 & \\
\phi_1\ U_{<\gamma}\ \phi_2 & : & \text{Compute}_{(U_<)}\ m\ \gamma\ \phi_1\ \phi_2 & \text{– temporal operators} \\
\phi_1\ S_{<\gamma}\ \phi_2 & : & \text{Compute}_{(S_<)}\ m\ \gamma\ \phi_1\ \phi_2 & \\
\eta_1 < \eta_2 & : & \text{Compute}_{(<)}\ m\ \eta_1\ \eta_2 & \text{– relational operator}
\end{array}
$$

  end
end

Algorithm 2: Computation of RMTL-$\int_3$ terms ($\text{Compute}_{(\eta)}$) and formulas ($\text{Compute}_{(\varphi)}$)

Operator $<$. Given two terms $\eta_1, \eta_2 \in \Gamma$, the formula $\eta_1 < \eta_2$ is evaluated relative to a model (κ, υ, t) by the function $\text{Compute}_{(<)} : (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \to \Gamma \to \Gamma \to \mathbb{B}_3$, also shown in Figure 3.2. The time complexity of this computation function depends on the time complexity of $\text{Compute}_{(\eta)}$ since any formula containing only the relation operator $<$ cannot have size greater than one, or consume any input symbols. For instance, a formula with depth two is $\int^1 \phi_1 < \int^1 \phi_1$, and with four is $\int^1(\int^1 \phi_1 < \int^1 \phi_1) < \int^1(\int^1 \phi_1 < \int^1 \phi_1)$.

Term $\int$. The evaluation of a duration term $\int^\eta \phi$ in the model (κ, υ, t) is performed by the function $\text{Compute}_{(\int)} : (\mathbf{K} \times \Upsilon) \to \mathbb{R}_{\geq 0} \to \mathbb{R} \to \Phi^3 \to \mathbf{D}$, again defined in Figure 3.2. It has linear time complexity in the size of the timed sequence, and constant time complexity in the formula size assuming that $\text{Compute}_{(\eta)}$ has constant complexity. $+$ and $\times$ terms are directly mapped into their respective computational operations. The complexity of those operations is directly related to the number of terms. Given a formula φ and a measure m_η describing the number of operators $+$ and $\times$ occurring in a formula φ, we have a linear lower bound of time complexity in $\mathcal{O}(2^{m_\eta})$ again assuming that $\text{Compute}_{(\eta)}$ has constant complexity.

Compute	Ω	Big-O
(α)	$\Omega(1)$	$\mathcal{O}(1)$
$(\int)$	$\Omega(n_\kappa - 1)$	$\mathcal{O}(n_\kappa)$
$(+), (\times)$	$\Omega(2^{m_\eta - 1})$	$\mathcal{O}(2^{m_\eta} - 1)$
(p)	$\Omega(1)$	$\mathcal{O}(1)$
$(\neg)$	$\Omega(m_\neg)$	$\mathcal{O}(m_\neg)$
$(\vee)$	$\Omega(2^{m_\vee - 1})$	$\mathcal{O}(2^{m_\vee})$
$(<)$	$\Omega(1)$	$\mathcal{O}(1)$
$(\mathrm{U}_<), (\mathrm{S}_<)$	$\Omega(1)$	$\mathcal{O}(2 \cdot n_k)$
$(\varphi), (\eta)$	$\Omega(2(n_\kappa)^2 \cdot (2^{m_\varphi} - 1) - 4(n_\kappa)^2 + n_\kappa \cdot (2^{m_\varphi} - 1) - 2(n_\kappa))$	$\mathcal{O}(\binom{k+m_\varphi}{m_\varphi} \cdot 2^k)$

Table 3.1: Complexity results of the Algorithm 2

Time complexity of the evaluation algorithm. We are now in a position to present a straightforward recursive top-level evaluation Algorithm 2 excluding *punctual* temporal operators, using the previous definitions for auxiliary computations. Let m_φ be a measure for $\vee$, $<$, temporal operators, and non-rigid terms. Given the complexity of these formulas and term operators, and knowing that all temporal operators have the same complexity as the until operator, we have by semantic definition that any combination of formulas has higher complexity. As such, the complexity of Algorithm 2 is exponential in the input size of the formula and the timed state sequence, as given by the upper bound identified above.

Table 3.1 summarizes the complexity for each individual evaluation function. For each function (α), Compute$_{(\int)}$, $(+)$, $(\times)$, (p), Compute$_{(\neg)}$, Compute$_{(\vee)}$, Compute$_{(<)}$, Compute$_{(U_<)}$ and Compute$_{(S_=)}$, we assume that the function Compute$_{(\varphi)}$ executes in constant time in order to identify the source of complexity for each case. This happens in the evaluation of $<$, $\int$, $+$ and $\times$. We also have asymptotically identified a lower bound for the complexity of the evaluation algorithm for each case, including Compute$_{(\eta)}$ and Compute$_{(\varphi)}$. Although the complexity is exponential, we have that in average the behavior may be much closer to the lower bound, as we will see in Chapter 5.

In order to analyze the space complexity of the synthesized monitors we first note that the synthesis algorithm produces monitors written using pure lambda functions. Following our approach, each formula ψ in RMTL-$\int_3$ to be synthesized, of length m_ψ, will originate a set of λ-expressions whose global size is in $\mathcal{O}(m_\psi)$, and whose mutual recursion pattern (or call graph) is free of cycles, since the invocations follow the structure of the formula ψ. Execution of these λ-expressions relies on a functional, stack-based mechanism, and it follows that the number of push/pop operations performed will be in $\mathcal{O}(m_\psi)$. The required

stack size will thus be linear in m_ψ, and *constant in the input trace size*. Therefore, the generated monitoring algorithms have *constant space complexity* regarding the trace size, as our experimental results will confirm in Chapter 5.

Summary

In this chapter we have presented two distinct synthesis approaches for the well-behaved fragment of MTL-$\int$. The approach based on SMT solvers is essential to prove some safety properties about the basis of the monitoring architectures, and the other approach can be an appropriate extension for checking more expressive and complex duration properties. This combination is essential to cover the nature of the duration properties since the majority of such properties are practically impossible to check statically. In this way, synthesis of monitors acts as a complement to cover unchecked properties and draw verdicts about the past executions. A three-valued extension of the RMTL-$\int$ formalism is also defined which allows us to carry out coherent sequential evaluation of traces.

As a final note, this work will be used as basis for the next chapter, where we address the problem of determining which properties can be discarded statically and which parts can be addressed at runtime in the context of real-time systems scenario.

Chapter 4

RV-RMTL-$\int$ Framework

RV methods can be applied to systems where the source code is not available , or in those cases where we have access to the code but the complexity of the system's requirements is too high to be addressed via any of the most commonly used static verification approaches. For RV, only a monitoring model needs be considered beforehand as well as the monitor synthesis mechanisms.

In this chapter, we introduce a component-based framework that helps us to manage the composition of the runtime monitors with the target system in order to support external observations of the system at execution time. It also ensures properties such as the *maximum detection delay* of the monitors, as well as the encoding of the scheduler behavior, which are features that are of paramount importance for hard real-time systems. In the remaining part of the chapter, we introduce the notion of *safe monitor* and describe a *domain specific language* (DSL) that supports the construction of different safe components and monitoring sketches.

4.1 Components

Before introducing components' types and the framework model itself, we will recall the preliminary definitions of a real-time task set, a periodic resource model, and an event sequence.

We will assume *task sets* $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, such that $n \in \mathbb{N}^+$ is the number of tasks $\tau_i = (p_i, e_i)$ where p_i and e_i are, respectively, the period and the worst-case execution time of τ_i. Each task $\tau_i \in \mathcal{T}$ is implicitly periodic and has implicit deadline. A *periodic resource model* ω is a tuple $(\mathcal{T}, \pi, \theta, rm)$, where $\mathcal{T} \subseteq \Gamma$, π is the *replenishment period*, θ is the *server budget*, and rm is the rate monotonic scheduling algorithm. The set of periodic

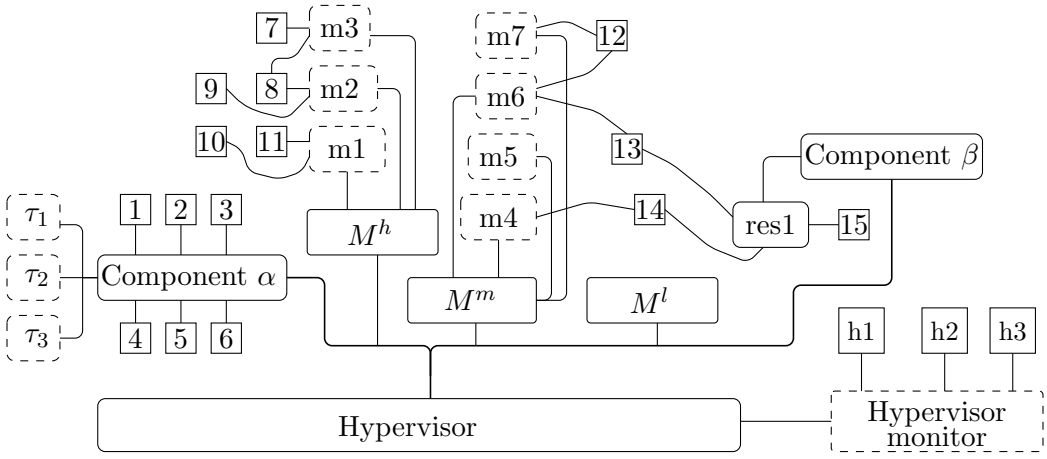


Figure 4.1: Component-based sketch with one hypervisor and quasi-omniscient monitors.

resource models is denoted by $\Omega = \{\omega_1, \omega_2, \ldots, \omega_m\}$ for an arbitrary $m \in \mathbb{N}^+$. We denote the index i of a task by τ_i and the index j of a resource by ω_j, where $0 < i \leq n$ and $0 < j \leq m$ holds, respectively. The outputs of a resource model ω are *sequences of events*.

Let us now consider the alphabet of events $\mathcal{E}$. Each element can be of one of the following types: a *task release event* RE; a *task start event* ST; a *task sleep event* SL; a *task resume event* RS; a *task stop event* SO; a *resource budget release event* RN; or a *general purpose event identifier tuple* EV. We also consider that general purpose events are special since they include a certain event identifier. Events can also have inheritance over other events as denoted by $e_{1(e_2)}$, for any $e_1, e_2 \in \mathcal{E}$. For short, we adopt the notation $e_{1(\omega_j,\tau_i)}$ that means that the event e_1 inherits from EV with event identifier tuple (ω_j, τ_i), for any $i, j \in \mathbb{N}$, $\omega_j \in \Omega$, and $\tau_i \in \mathcal{T}$.

Event sequences are a formalism that allows us to describe the scheduler behavior, creating a generic event language that a system can produce. If a system produces unexpected event words, we shall consider it a faulty system. Similar meaning is also established for temporal logic observations [Lakhnech and Hooman, 1995]. A sequence of events, also known as *execution trace*, is an infinite sequence

$$\rho = (e_1, t_1)(e_2, t_2) \cdots$$

of time-stamped events (e_i, t_i) with $e_i \in \mathcal{E}$ and $t_i \in \mathbb{R}^+$. The sequence satisfies monotonicity and progresses, *i.e.*, $t_i \leq t_{i+1}$ for all $i \in \mathbb{N}^+$, and for all $t \in \mathbb{R}^+$ there is some $i > 0$ such that $t_i > t$, respectively.

After having introduced these preliminary definitions, we are able to start describing the *compositional monitoring framework* (CMF). This framework is composed from a set of components of one of the following types:

- (Timing Constraint) A timing constraint Ψ is a set of constrained temporal formulas in RMTL-$\int_3$.
- (Task) A task *tsk* is a pair (τ_1, Ψ) such that $\tau_1 \in \Gamma$ and Ψ are constrained formulas encoding several task behaviors to be checked at runtime.
- (Resource) A resource *res* is a tuple of the form (ω, Ψ), where $\omega \in \Omega$ is a resource model, and Ψ is a set of constrained formulas to be checked at runtime.

We assume the existence of a relation for the composition of resources, tasks and constraints. This relation is restricted by the way that components are composed with other components of the same type. Let us now introduce a small practical example of a two-level hierarchy system to be used along this chapter.

Example 10. *Consider the Figure 4.1 as a component-based graphical model where each link connecting point A to point B means "A relates with B". Solid boxes are resources, dashed boxes are tasks, and squared solid boxes are formulas in RMTL-$\int_3$. These formulas will be automatically synthesized with respect to a given monitoring model and some properties such as if the maximum detection delay of the monitors will be ensured by the framework.*

In this sketch, we also have distinct resources M^h, M^m and M^l which encapsulate monitors by priority based on different criticality levels. This allows us to identify until what point this framework can deal with elastic executions. By elastic execution we mean a system composed by several resources that can use different budgets over different time instants (a feature that we will describe in the use case presented in the next chapter). A hypervisor is no more than a component that only exists in this sketch for encapsulation purposes. This component contains a set of quasi-omniscient monitors (resp. hypervisor monitors) that reach verdicts about the assumptions of the monitoring architecture (a notion of monitors' hierarchy as described in the end of this chapter).

Intuitively, we have presented the purpose of CMF through this example, i.e., as a framework to deal with description of the monitoring sketches and also to split the properties to be checked statically and dynamically. Note that *task* and *resource* components are simple encodings of task and resource model behaviors coupled with *timing constraints* that are encoded as RMTL-$\int_3$ formulas to be safely monitored. Our major goal is to ensure that every monitor complies with the expected *maximum detection delay*, since *worst case execution time* (WCET) violations of one or more tasks may interfere with each other and also other non monitoring tasks, resulting in an undesirable environment.

In addition, the predictability of our framework with respect to the event sequences can be established by identifying the relevant or critical events, and preserving the partial order

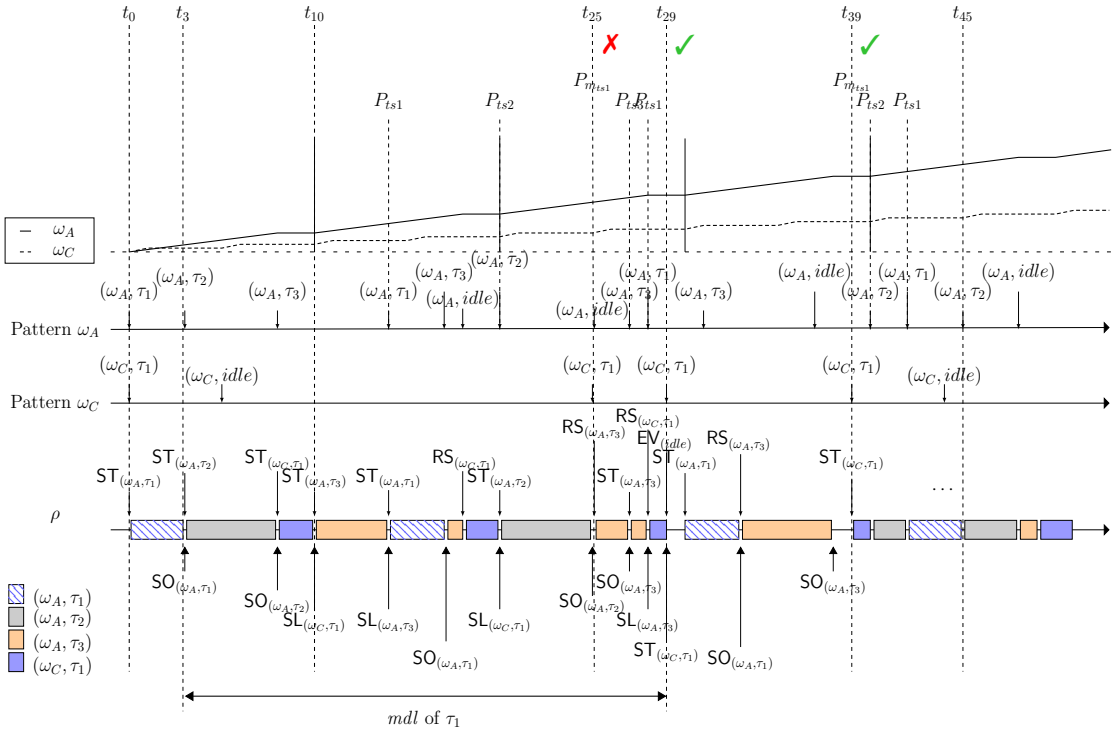


Figure 4.2: Example of patterns and the global trace generated by the composition of resource models defined in the Example 11

of events arrival for monitor processes. We need to save this order due the possibility of using more than one trace/buffer in the same sketch of this framework. We also identify the event SO as the critical event for schedulability analysis, since it is the event triggered when a task job finishes its execution. We denote the critical events by the subset $\mathcal{E}_{cr} \subseteq \mathcal{E}$, the prefix-tree which preserves the partial order of events for all possible executions by pt, and the maximum duration of a prefix trace by s. Given these predictable traces pt, we are able to evaluate the response time of the monitor m for each trace $\rho \in pt$ using the formula

$$\Box_{<s} \bigwedge_{e \in \mathcal{E}_{cr}} e \rightarrow \Diamond_{\leq \gamma} \mathsf{SO}_{(e,m)}, \tag{4.1}$$

where $\mathsf{SO}_{(e,m)}$ is the triggered event that the monitor m generates at the end of its complete execution for monitoring the task/resource that has been triggered the event e, and s is the time window to be considered.

Example 11. *Let us assume two resource models ω_A with parameters ($\pi = 10,\ \theta = 8$) and ω_C with ($\pi = 5,\ \theta = 1$) described in Figure 4.2 containing three tasks $\tau_1(p = 14, e = 3)$, $\tau_2(p = 20, e = 5)$, and $\tau_3(p = 27, e = 7)$, and one task $\tau_1(p = 33, e = 4)$, respectively. We could see that to guarantee the maximum detection delay of the monitor task τ_1 in ω_C, the trace depicted in the Figure 4.2 needs to be generated. For the generation of this trace, we assume the well known* critical instant theorem *to find the worst execution trace as well*

as the hyper-period of the resource model to define the maximum length of the trace [Liu and Layland, 1973]. *Replacing the event* $\mathsf{SO}_{(e,m)}$ *with* $\mathsf{SO}_{(\omega_C,\tau_1)}$ *in Formulae 4.1, we are able to check the maximum detection delay of our trace, which corresponds to a value* γ *greater than* 26 *time units depending on the desired WCET, and where the instants* t_{29} *and* t_{39} *exemplify the allowed periods. In this case, the maximum detection delay may increase depending on the monitor period when greater than* 42 *time units. Note that this example only works for the assumption of the critical instant theorem and/or the consequent enumeration of the possible traces, and therefore is not general enough.*

In the remaining part of this chapter, we illustrate how to overcome this issue in an elegant way without the assumption of prefix trees or the critical instant theorem by reformulating the time constraints check into a *satisfiability problem*. Without enumerating every possible trace or selecting the worst trace, which is impossible in a multi-processor setting due essentially to *anomalies* [Andersson and Jonsson, 2002], we are able to specify and analyze schedulability of multi-processor systems, notably the ones with *dependent tasks*.

4.2 Formal Specification of Periodic Resources

To simplify the expressions' encoding of the safe CMF model, we first introduce some syntactical notations and formula abbreviations.

The set of tasks with higher-priority (and including) than τ_i for ω_j is denoted by $\gamma_{\omega_j}^{\tau_i}$. We also use h as the hyper-period, and the operator $T \triangleq true$ as T defining a shorthand for $true$. For events, we adopt the following notations: $\mathsf{EV}(\omega_j, \cdot)$ denotes the set of events that can be generated by the resource model ω; $\mathsf{EV}(\omega_j, \tau_i)$ denotes the set of events that can be generated by the task τ_i in the resource model ω_j; $evs^+(\omega_j, \tau_i)$ is defined by

$$evs(\omega_j, \tau_i) \vee \mathsf{SO}_{(\omega_j,\tau_i)} \vee \mathsf{EV}(\omega_j, \tau_i) \vee \mathsf{RE}_{(\omega_j,\tau_i)},$$

with $evs(\omega_j, \tau_i)$ defined by

$$\mathsf{ST}_{(\omega_j,\tau_i)} \vee \mathsf{RS}_{(\omega_j,\tau_i)} \vee \mathsf{RN}_{(\omega_j)},$$

which specifies all events that a task τ_i in the resource model ω_j can trigger; $evs^-(\omega_j, \tau_i)$ denotes the formula resulting from the removal of the $\mathsf{RE}_{(\omega_j,\tau_i)}$ and $\mathsf{SO}_{(\omega_j,\tau_i)}$ events from $evs^+(\omega_j, \tau_i)$; finally, $evs^*(\omega_j, \tau_i)$ denotes the formula resulting from the removal of the $\mathsf{ST}_{(\omega_j,\tau_i)}$ and $\mathsf{SO}_{(\omega_j,\tau_i)}$ events from $evs^+(\omega_j, \tau_i)$.

A resource component $(\omega_j, \{\psi_1, \psi_2, ...\})$ is made of the set of formulas $\{\psi_1, \psi_2, ...\} \subset \Phi^3$ that will be automatically synthesized as a collection of online monitors, and a resource

model ω_j that captures the semantic nature of the resource with a formula containing properties such as the resource model budget supply, the schedulability policy, the task set durations and period, and other intrinsic settings for complete specification of the component. Φ^3 is a set of three valued formulas as defined before, and the binary operator $\varphi_1 \ominus \varphi_2$, meaning *next implies*, is a shorthand for $\varphi_1 \rightarrow (\varphi_1 \,\mathrm{U}_{<b}\, \varphi_2)$, where b is a fixed and sufficiently large number.

The resource model budget supply is specified by the formula

$$\Box_{\leq h}\, \mathsf{RN}_{(\omega_j)} \ominus \left(\Diamond_{=\pi}\, \mathsf{RN}_{(\omega_j)}\right) \wedge \int^{\pi} \bigvee_{\tau_i \in \tau} evs^{+}(\omega_j, \tau_i) \leq \theta, \tag{4.2}$$

where ω_j is one resource model, π and θ are their renewal period and budget, and $\mathsf{RN}_{(\omega_j)}$ is the budget renewal event. This formula states that for each occurrence of the event $\mathsf{RN}_{(\omega_j)}$ in the resource model ω_j, the duration of the other events until π time units does not overpasses the budget θ per period π.

For the partial order of the task releases, as defined by the scheduler policy rm, we introduce the RMTL-$\int_3$ formula

$$\Box_{\leq h} \bigwedge_{\tau_i \in \mathcal{T}} \left(\mathsf{RE}_{(\omega_j,\tau_i)} \ominus \left(ev(\omega_j, \tau_i)\ \mathrm{U}_{\leq p_i}\ \mathsf{SO}_{(\omega_j,\tau_i)}\right)\right), \tag{4.3}$$

where

$$ev(\omega_j, \tau_i) \triangleq \left(\bigvee_{\tau_k \in \gamma_{\omega_j}^{(\tau_{i-1})}} evs^{+}(\omega_j, \tau_k) \right) \vee evs^{-}(\omega_j, \tau_i)$$

and $\gamma_{\omega_j}^{(\tau_{i-1})}$ denotes the set of higher-priority tasks, excluding events triggered by the task τ_i. This formula means that for every event $\mathsf{RE}_{(\omega_j,\tau_i)}$ there is always an event $\mathsf{SO}_{(\omega_j,\tau_i)}$, and that the events occuring before $\mathsf{SO}_{(\omega_j,\tau_i)}$ should be any event from τ_i's higher-priority tasks.

The duration of tasks allocated to one resource model is specified by the formula

$$\Box_{\leq h} \bigwedge_{\tau_i \in \mathcal{T}} \mathsf{RE}_{(\omega_j,\tau_i)} \ominus \int^{p_i} \bigvee_{\tau_k \in \gamma_{\omega_j}^{(\tau_i)}} evs^{+}(\omega_j, \tau_k) \leq\ e_i. \tag{4.4}$$

Note that the $\leq$ operator should be changed to $\geq$ in order to specify the absolute WCET of the task set.

We also specify other properties such as the precedence of the event $\mathsf{SO}_{(\omega_j,\tau_i)}$ (*i.e.*, each event $\mathsf{ST}_{(\omega_j,\tau_i)}$ may be followed by an event $\mathsf{SO}_{(\omega_j,\tau_i)}$, but the event $\mathsf{SO}_{(\omega_j,\tau_i)}$ occurs since $\mathsf{ST}_{(\omega_j,\tau_i)}$ occurs). The precedence of the event $\mathsf{SO}_{(\omega_j,\tau_i)}$ is specified by the formula

$$\Box_{\leq h} \bigwedge_{\tau_i \in \mathcal{T}} \mathsf{SO}_{(\omega_j,\tau_i)} \ominus \left(es(\omega_j, \tau_i)\ \mathrm{S}_{\leq p_i}\ \mathsf{ST}_{(\omega_j,\tau_i)}\right), \tag{4.5}$$

where

$$es(\omega_j, \tau_i) \triangleq \left(\bigvee_{\tau_k \in \gamma_{\omega_j}^{(\tau_{i-1})}} evs^{+}(\omega_j, \tau_k) \right) \vee evs^{*}(\omega_j, \tau_i).$$

The complete encoding of the component is given by the conjunction of the formulas 4.2, 4.3, 4.4 and 4.5. For the remaining part of the chapter, we define it by $\mathsf{PRM}(\omega_j)$, where ω_j is indexed according to certain workload parameters, allowing us to unroll the sub-formulas in the correct way. This partially concludes the formalization of the periodic resource model's behavior using RMTL-$\int_3$.

Note that in the Section 4.3 we will return to the hierarchical composition of the presented resource specification, but only after extending the formalization to *dependent tasks*.

4.2.1 Extension for dependent tasks

Adding dependence task checking is as easy as adding more timing constraint formulas. Properties such as "the dependent task (B) cannot begin until the task (A) completes" can be ensured as result of A being a pre condition for the result of B. Note that this is necessarily a more expressive model of dependent tasks than the ones presented in the literature [Goossens et al., 2016, Puffitsch et al., 2015, Baro et al., 2012]. Assuming that tasks are divided into several sections according to their flow graphs, we could specify that a section of a task has a dependence relatively to other tasks' sections. And, other constraints written in RMTL-$\int_3$ restricting other resources such as memory and network message passing can be asserted as well. It turns out that extending the model is modular, unlike the classical schedulability analysis tests where we may have to redo everything from scratch.

Example 12. *Let us take τ_1 as a system task and τ_2 a monitoring task, where each one executes in isolation in the resources ω_1 and ω_2. Consider the resources with the event control graph described in the Figure 4.3. The monitoring task has an arbitrary period and may contain two sub-events such as EV_1 and EV_2, or even execute arbitrarily. For the former case, these points are when the monitor contains enough/required symbols to consume, identified by the formula's morphology. Then, executing before these points does not make sense since it is wasting time and increasing pessimism in the schedulability analysis. EV_1 shall execute after $\mathsf{EV}_{(E)}$, and $\mathsf{EV}_{(H)}$ and EV_1 shall execute after $\mathsf{EV}_{(H)}$ and $\mathsf{EV}_{(C)}$. For the latter case, arbitrary execution incurs in executing the monitor before and after task τ_1 terminates, which in the worst case indicates that we need to execute the monitor after $\mathsf{SO}_{\omega_1,\tau_1}$ occurs. Executing along the system task is not safe, context-switches*

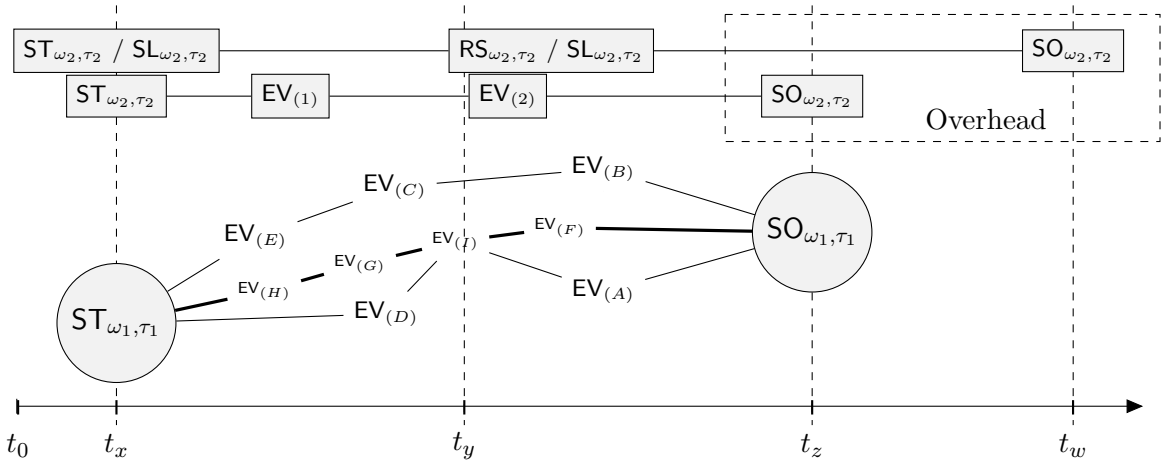


Figure 4.3: Flow graph of the scenario considered in Example 12 and 18

for resuming and entering the sleep state are unnecessarily required, and the overhead is $t_w - t_z$.

Indeed,this is a generalization for elasticity of budgets and periods along execution time of resources. Offsets can also be applied for starting monitor execution, avoiding $\mathsf{ST}(\omega_2, \tau_2)$ *at time* t_x*, and only the hyper period among different cores is required to encode schedulability of multi-core systems in a satisfiability problem.*

Our approach is modular in the sense that it can be extended with minor efforts. It also allows us to manage sets of polynomial inequalities as in common real-time approaches and an hybrid between both formalizations may be an option. However, the drawback of the approach is that solving the generated problems in a practicable way may be challenging, a discussion that will take place in the next chapter. Note also that SMT solvers have been the target of significant advances in the last years, and heuristic approaches proposed in hard real-time systems literature fail to deal with this type of extension since they behave badly with non local properties [Puffitsch et al., 2015]. Currently, only *linear programming* and *constraint programming* techniques are successfully applied to solve parts of this problem for a high number of tasks and several working cores without any proof generation. When using SMT solvers the same does not happen.

To the best of our knowledge, there are presently no published works in the RV literature that, instead of considering a unique period for a monitor, consider multiple periods for the same monitor, each one activated at certain time instants. This is a pattern of periods which we will call *monitoring with elastic execution*. Periods and execution times are not fixed. We know that event-driven approaches are not so feasible for embedded systems and even less feasible for hard real-time systems where predictability and timing correctness are required [Medhat et al., 2015, Bonakdarpour et al., 2013]. Commonly,

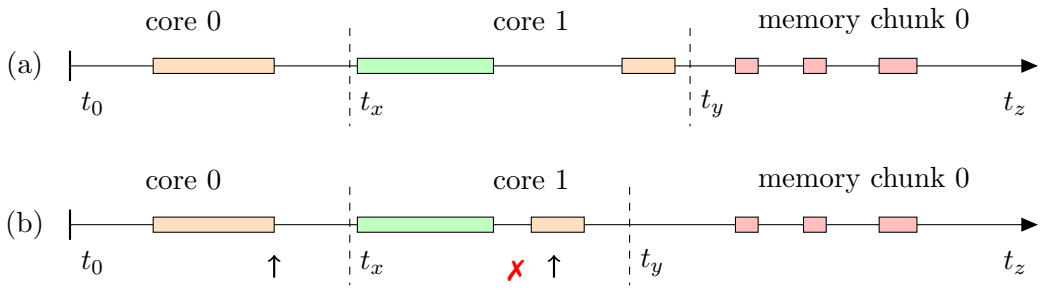


Figure 4.4: Encoding of processor mapping and memory mapping

such approaches propose finding new parameters for existing scheduling algorithms. Time-triggered approaches are too generic but predictable in comparison with the event-driven ones. Moreover, the classical schedulability analysis can be readily applied but they are in general too pessimistic for RV of hard real-time systems. This is the novelty of our approach, that instead of being too generic allows us to define more constraints about the execution of the monitors, including the extension for multi-core systems. Proofs are also generated for each sketch and we only need to assume the synthesis steps.

Example 13. *Consider that we have to get a scheduler for dependent tasks executing in a multi-core system. Given our approach we can deal with it by simply extending the formulas as easily as constructing a formula of the form*

$$\bigwedge_{\omega \in \Omega} \Diamond_{=lcm(\omega)} \; PRM(\omega),$$

where lcm is a function returning the least common multiple for a resource ω. We are assuming that each resource ω executes in different cores.

In Figure 4.4, we have a graphical representation of this encoding, including the way we reserve memory. lcm will give us 0 or t_x. t_y is used to find that

$$\Diamond_{=t_y} \int^{t_z} \mathsf{EV}_{(\omega,\mathsf{usage})} < 10,$$

which means that the memory usage should be less than 10 space units. Note that we reason about both space and time in the same trace. In case of trace (b), the overlapping of the same execution unit when migrating to different cores is not allowed. From these, we know where the task allocates its stack ensuring that it is allowed by the specification. We specify it by the formula

$$\Box_{<t_x} \; \left(\mathsf{ST}_{(\omega,\cdot)} \vee \mathsf{RS}_{(\omega,\cdot)}\right) \rightarrow \neg \Diamond_{=t_x} \; \mathsf{ST}_{(\omega,\cdot)} \vee \mathsf{RS}_{(\omega,\cdot)},$$

for any resource $\omega \in \Omega$. We see a task only making use of a local stack, indicating that the memory allocation is predictable. By stack we mean a portion of memory allocated continuously and dedicated only to a task.

Another feature that we have is the position at which the local stacks are allocated. Instead of providing an inequality such as $\sum_{\tau_i \in \tau} size(\tau_i) < L$*, where we do not know anything about the allocation as in [Puffitsch et al., 2015], we ensure that there is enough space and the order of the allocation. Knowing the portion where we allocate memory can help us to speedup the execution of the system since we may have non homogeneous memories in the system, which means different speed accesses.*

This example has illustrated how multi-processor scheduling can be encoded by simply extending the presented formalization, including dependent tasks.

Another feature that we might refer here is how *contention* accessing a shared memory resource can affect the schedulability analysis, as has been exemplified in the Figure 4.4. Instead of getting worst-case bound on the contention, we can formulate this constraint in a different way. The presented approach avoids considering a possible pessimistic worst-case scenario of contention a priory. For instance, if two cores access the same region of memory then this will cause contention somehow. However, if we enforce these cores to use the memory in a different time instant or during better circumstances then the contention is relaxed, and the worst case will not be worth applying. In this way, this approach using temporal formulas describing temporal patterns may be more appropriate in terms of access patterns to reduce contention, improving on the techniques that can be found in the literature.

Scrutinizing the importance of WCET and dependent constraints in monitoring. Let us now see an important case that is often neglected in the RV literature. WCET has been commonly assumed for constructing schedulability analysis of different schedulability algorithms. However, this introduces some issues regarding the pessimism and the practical application of these approaches for analysis of runtime monitors. WCET is a general assumption that is sufficient for cases where there are no dependency constraints on tasks or resources, i.e., they are independent or partially independent.

Consider a system with a taskset containing a task where WCET depends on the execution of the other tasks as in the running Example 11. It turns out that the schedulability of the taskset is infeasible according to [Shin and Lee, 2008], since the WCET may tend to be unachievable and/or too pessimistic to be considered, and even due to this test only working with independent tasks (i.e., unsafe for our purpose). By assuming a simple dependency constraint, we may find a lower WCET and a schedulable taskset using our logic fragment. This is how monitors can behave if they are depending on timing constraints, and properties such as maximum detection delay are necessary for ensuring it.

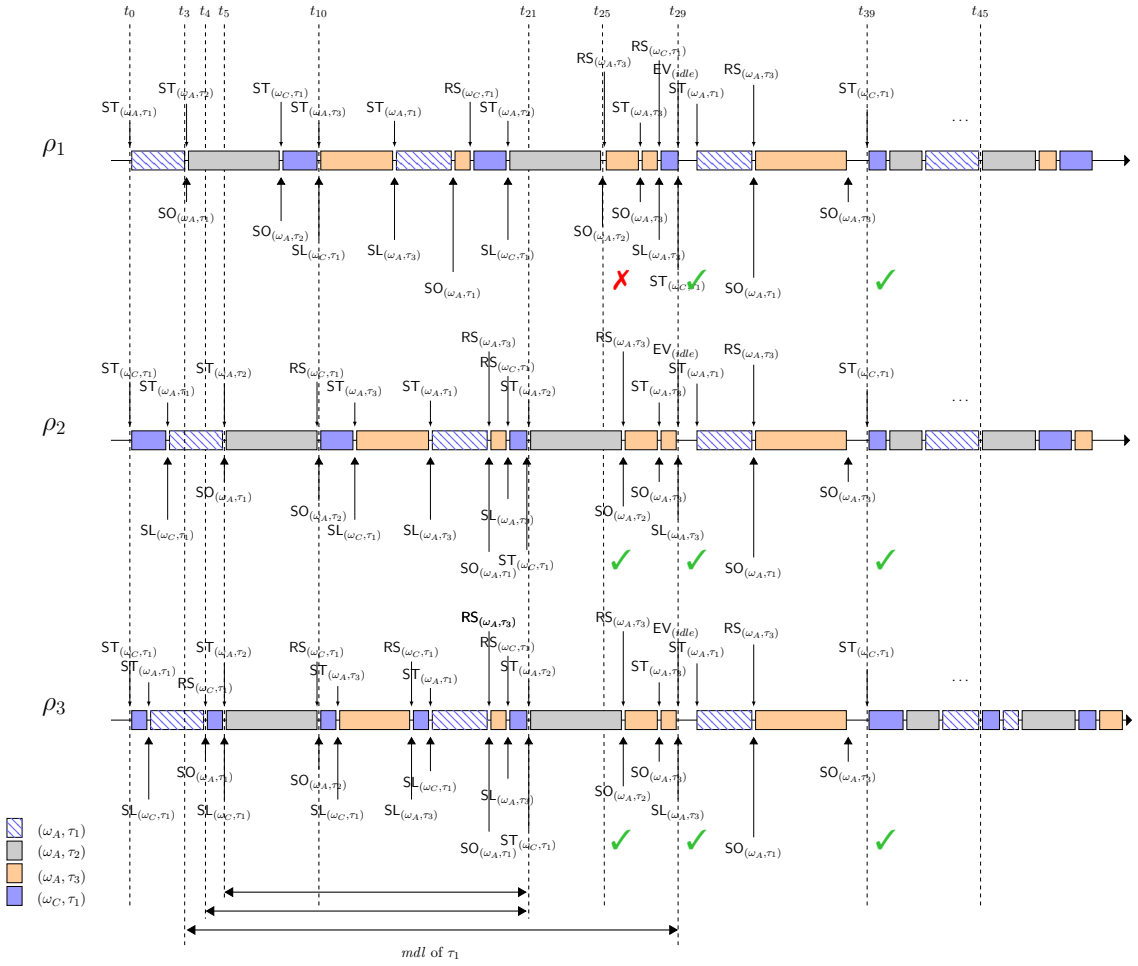


Figure 4.5: Diagram with evidences of infeasibility

The Figure 4.5 provides evidences of different traces ρ_1, ρ_2, and ρ_3, where using periodic resource models can introduce such infeasibility. ρ_2 and ρ_3 are traces where the maximum detection delay is lower, but they have more context-switches, and portions of execution where the task may be wasting time. ρ_1 is an acceptable trace, however, ρ_2 and ρ_3 may not work due to discharging 2 and 1 time units before executing τ_1. This time may be crucial for executing a monitor for τ_1 under the assumption of the WCET of the task. More precisely, a monitor task will execute as the system provides symbols for consumption and the first block of both traces will not be considered in these traces.

To provide a real WCET for this application without requiring to largely estimate it, we only need to find the exact WCET of a job with the assumption that each entry of this job will be executed when events are ready to be consumed, using a time triggered approach. For that, we need to statically assert the formula 4.1 and a formula encoding their precedences at the level of the internal events of a task or a set of tasks.

RV of explicit time is inherently dependent of past execution and as such we need to adopt

models such as the ones containing dependent tasks with exact schedulability to avoid pessimism. Monitoring and pessimism do not combine, since the goal of a monitor is to interfere as low as possible in the system but increase as high as possible the reliability of the system.

4.3 Safe Components and Monitors

In this section, we will continue extending the scheduling formalization of resource models in order to support construction of safe components and monitors using RMTL-$\int_3$. We consider *mdl* as the function transforming a formula to be monitored into one formula including the *maximum detection delay* assertion.

Let us recall that $\varkappa_i$ is a prefix of a timed sequence κ at i, and κ_i is a suffix of κ at i. We write $\kappa \models P$ when the time sequence κ satisfies the property P.

Let us start by defining what is meant by a safety property [Alpern and Schneider, 1987].

Definition 19. Let K be the set of infinite timed sequences, and P a property. P is a safety property *iff* for all $\kappa \in K$ such that $\kappa \not\models P$ there exists an $i, i \leq 0$ such that for all $\kappa_b \in K$, $\varkappa_i \kappa_b \not\models P$.

Since monitoring a property does not ensure anything by itself, we need to establish the following propositions.

Proposition 1. *Let ϕ be a monitoring formula in RMTL-$\int_3$. The monitor formula ϕ is safe iff the formula $mdl(\phi)$ is satisfiable.*

Proof (sketch). Consider that ϕ is a safety property, and *mdl* constructs the set E of sub-formulas from ϕ. Then, we have to prove that the formula $\Box_{<a} \bigwedge_{e \in E} \left(e \rightarrow \Diamond_{\leq \gamma} \mathsf{SO}_{(m)}\right)$ is safe. Since for all $e \in E$, e is a safe formula, it remains to prove that $\Box_{<a} \left((\neg e_1) \vee \Diamond_{\leq \gamma} \mathsf{SO}_{(m)}\right) \wedge \left((\neg e_2) \vee \Diamond_{\leq \gamma} \mathsf{SO}_{(m)}\right) \ldots$ is a safe formula. The proof follows by Definition 19 for the cases $\neg e, e_1 \vee e_2$, and $e_1 \, \mathrm{U}_{<\gamma} \, e_2$, which we omit here for simplicity. Hence, if it is satisfiable then we have a safe monitor. □

Proposition 2. *Let C be a component of the form $(\Gamma, \omega, \vartheta, \Phi_{sub})$, and Φ_{sub} is equal to $\phi_1, \phi_2, \ldots, \phi_n$ for an arbitrary length n. The component C is safe iff the formula $\mathsf{PRM}(\omega) \wedge \bigwedge_{i=1}^{n} mdl(\phi_i)$ is satisfiable.*

Proof (sketch). Assuming that $\mathsf{PRM}(\omega)$ is a safe formula, the proof follows directly from Proposition 1. □

Lemma 10. *Let C_1 and C_2 be two components of the form $(\Gamma_1, \omega_1, \vartheta_1, \Phi_1)$ and $(\Gamma_2, \omega_2, \vartheta_2, \Phi_2)$ where Φ_{sub} is equal to $\Phi_1 \cup \Phi_2$ and of the form $\phi_1, \phi_2, \ldots, \phi_n$ for an arbitrary length n. Arbitrary execution of C_1 and C_2 is safe iff the formula $\mathsf{PRM}(\omega_1) \wedge \mathsf{PRM}(\omega_2) \wedge \bigwedge_{i=1}^{n} mdl(\phi_i)$ is satisfiable.*

Proof (sketch). The proof follows directly from Proposition 2 for C_1 and C_2. □

Let us now go back to the Example 10 containing an hierarchy of monitors. A hierarchy of components as described in Figure 4.1 can be specified based on arbitrary execution of components.

The composition for the case of the hypervisor of the form $(\Omega, \eta_p, \eta_m, \phi_h)$, where Ω is a set of resource models, η_p a set of processors, and η_m a set of memories, is indeed a composition of the components inside Ω and $\bigwedge_{\phi \in \phi_h} mdl(\phi)$.

Ensuring the safety property for each monitoring formula is of extreme importance in order to ensure that nothing bad happens when other monitors and system' tasks are combined. To facilitate the description of monitoring schemes using a more natural language for program developers, we will introduce next a micro resource DSL. Note that every construction of this DSL is on top of the presented formalization of the last sections.

4.4 DSL for components

Regarding resources, tasks, and other abstractions for task jobs and execution units of RTS, there are no DSLs appropriate to reason about resource availability and schedulability. In this section we introduce the μDSL language that have been designed to appropriately deal with resources and tasks among other constraints such as describing functional properties, including safety and liveness properties. Let us now introduce the syntax and establish how this language is synthesized to RMTL-$\int_3$ by the respective operational semantics.

Definition 20 (Syntax)**.** Let op_{tk} denote one of the operators $\succ$ or $\bowtie$, where $\succ$ means the relation of the priority of tasks, and $\bowtie$ means that two tasks can be executed with the same priority, or execute arbitrarily. The operator for resources is $op_{rs} \in \{\|, \gg\}$, where $\|$ means that the resources execute in parallel, and $\gg$ means that the resources have a priority relation. We introduce a mapping operator $\overset{m}{\mapsto}$ for constraining the resources to memory regions. For instance, the expression $res(tsk(10,3),5,10) \overset{m}{\mapsto} chk(1)$ means that the resource $res(tsk(10,3),5,10)$ is mapped to the first chunk of memory. In a similar way, we use the operator $\overset{c}{\mapsto}$ for mapping resources to cores. For instance, the expression

$res(tsk(10,3),5,10) \overset{c}{\mapsto} cre(1)$ means that the resource will be executed in core one. We also define chk as intervals (e.g, $[a,b[$, $a,b \in \mathbb{N}^+$) mapped to memory chunks, and cre as a map of core indexes to Booleans. Finally, we define ct as a shallow translation of RMTL-$\int_3$ to express the same timing constraints. The μDSL is inductively defined by task expressions tk and resource expressions rs, as follows:

$$
\begin{aligned}
tm &::= vl \mid [ct]_v \\
ct &::= ev \mid \neg ct \mid ct_1 \vee ct_2 \mid ct_1 \wedge ct_2 \mid ct_1 \rightarrow ct_2 \mid ct_1 \twoheadrightarrow ct_2 \mid [[ct]]_{tm} \mid tm_1 < tm_2 \\
tk &::= tsk(p,e) \mid tk_1 \; op_{tk} \; tk_2 \\
rs &::= res(tk,\pi,\omega) \mid rs_1 \; op_{rs} \; rs_2 \mid rs \overset{m}{\mapsto} chk \mid rs \overset{c}{\mapsto} cre \mid rs \triangleleft ct
\end{aligned}
$$

where $tsk(p,e)$ is a task identified by a period $e \in \mathbb{N}^+$ and an *execution time* $e \in \mathbb{N}^+$, and $res(tk,\pi,\theta)$ is a resource with period $\pi \in \mathbb{N}^+$ and budget $\theta \in \mathbb{N}^+$.

Definition 21 (Operational semantics)**.** The semantics of our μDSL will be given by a set of rules having as premises and conclusion judgments of the form $\langle a, \Phi \rangle \Rightarrow \langle b, \Phi' \rangle$ with the meaning that a reduces to b and the current formula Φ being synthesized is updated to Φ'. Note that this is a small step semantics.

The compositional semantic rules as well as the complementary rules are defined in the Figure 4.6. The semantic rules for expressions using $\mapsto$ and $\triangleleft$ operators are also included. Note also that the remaining rules for reducing ct are a shallow translation of RMTL-$\int_3$, and no modifications in the syntax of the logic occurs. $[ct]_v$ is the same as $\int^v ct$, $ct_1 \twoheadrightarrow ct_2$ is the same as $ct_1 \, \mathrm{U}_{\sim b} \, ct_2$ with b sufficiently large, and $[[ct]]_{tm}$ is the same as $ct \wedge [ct]_{tm}$.

Let us now consider the events defined above in this chapter, and the identifier " $'$ " for labeling sub-formulas. Remark also that terminal rules cpl_2 and cpl_3 make changes according to the formal specification introduced in Section 4.2 for resources and tasks, respectively. chk, cre and ct rules are used for mere labeling.

We exemplify now two options that can be adopted. The first option is defining one formula generated by unfolding the temporal formula until a desired time bound. For instance, considering the punctual formula, which may be impractical for larger bounds. For the second option we need the definition of an invariant with a built-in implication, since we do not require to be constantly evaluating the until operator for each time instant, but only at certain time instants. In this case, the drawback is the definition of an auxiliary sub-formula, describing that an event is triggered once at each desired period.

Example 14. *Let us assume the expression* $tsk(9,3)$ *and the formula* ψ *equals to*

$$\big(\mathsf{ST}_{(\tau_1)} \vee \mathsf{RS}_{(\tau_1)} \vee \mathsf{SL}_{(\tau_1)} \; U_{<9} \; \mathsf{SO}_{(\tau_1)}\big) \wedge \int^{9} \mathsf{ST}_{(\tau_1)} \vee \mathsf{RS}_{(\tau_1)} \vee \mathsf{SL}_{(\tau_1)} \vee \mathsf{SO}_{(\tau_1)} < 3.$$

Composition rules

$$\text{cmp}_1: \frac{\langle tsk(a,b), \Phi\rangle \Rightarrow \langle ., \Phi''\rangle \quad \langle tk, \Phi''\rangle \Rightarrow \langle tk', \Phi'\rangle}{\langle tsk(a,b) \succ tk, \Phi\rangle \Rightarrow \langle tk', \Phi'\rangle}$$

$$\text{cmp}_{2_1}: \frac{\langle tsk(a,b), \Phi\rangle \Rightarrow \langle ., \Phi''\rangle \quad \langle tk, \Phi''\rangle \Rightarrow \langle tk', \Phi'\rangle}{\langle tsk(a,b) \bowtie tk, \Phi\rangle \Rightarrow \langle tk', \Phi'\rangle} \qquad \text{cmp}_{2_2}: \frac{\langle tsk(a,b), \Phi\rangle \Rightarrow \langle ., \Phi''\rangle \quad \langle tk, \Phi''\rangle \Rightarrow \langle tk', \Phi'\rangle}{\langle tk \bowtie tsk(a,b), \Phi\rangle \Rightarrow \langle tk', \Phi'\rangle}$$

$$\text{cmp}_3: \frac{\langle res(a,b,c), \Phi\rangle \Rightarrow \langle ., \Phi''\rangle \quad \langle rs, \Phi''\rangle \Rightarrow \langle rs', \Phi'\rangle}{\langle res(a,b,c) \gg rs, \Phi\rangle \Rightarrow \langle rs', \Phi'\rangle}$$

$$\text{cmp}_{4_1}: \frac{\langle res(a,b,c), \Phi\rangle \Rightarrow \langle ., \Phi''\rangle \quad \langle rs, \Phi''\rangle \Rightarrow \langle rs', \Phi'\rangle}{\langle res(a,b,c) \parallel rs, \Phi\rangle \Rightarrow \langle rs', \Phi'\rangle} \qquad \text{cmp}_{4_2}: \frac{\langle res(a,b,c), \Phi\rangle \Rightarrow \langle ., \Phi''\rangle \quad \langle rs, \Phi''\rangle \Rightarrow \langle rs', \Phi'\rangle}{\langle rs \parallel res(a,b,c), \Phi\rangle \Rightarrow \langle rs', \Phi'\rangle}$$

Complementary rules

$$\text{rsct}: \frac{\langle rs, \Phi\rangle \Rightarrow \langle ., \Phi''\rangle \quad \langle ct, \Phi\rangle \Rightarrow \langle ., \Phi'\rangle}{\langle rs \lhd ct, \Phi\rangle \Rightarrow \langle ., \Phi' \cup \Phi''\rangle}$$

$$\text{rschk}: \frac{\langle rs, \Phi\rangle \Rightarrow \langle ., \Phi''\rangle \quad \langle chk, \Phi''\rangle \Rightarrow \langle ., \Phi'\rangle}{\langle rs \overset{m}{\mapsto} chk, \Phi\rangle \Rightarrow \langle ., \Phi'\rangle} \qquad \text{rscre}: \frac{\langle rs, \Phi\rangle \Rightarrow \langle ., \Phi''\rangle \quad \langle cre, \Phi''\rangle \Rightarrow \langle ., \Phi'\rangle}{\langle rs \overset{c}{\mapsto} cre, \Phi\rangle \Rightarrow \langle ., \Phi'\rangle}$$

$$\text{chk}: \frac{}{\langle chk, \Phi\rangle \Rightarrow \langle ., \Phi'\rangle} \qquad \text{cre}: \frac{}{\langle cre, \Phi\rangle \Rightarrow \langle ., \Phi'\rangle}$$

$$\text{ct}: \frac{}{\langle ct, \Phi\rangle \Rightarrow \langle ., \Phi'\rangle}$$

$$\text{cpl}_1: \frac{\langle tk, \Phi\rangle \Rightarrow \langle ., \Phi''\rangle \quad \langle res(., a, b), \Phi''\rangle \Rightarrow \langle ., \Phi'\rangle}{\langle res(tk, a, b), \Phi\rangle \Rightarrow \langle ., \Phi'\rangle}$$

$$\text{cpl}_2: \frac{}{\langle tsk(a,b), \Phi\rangle \Rightarrow \langle ., \Phi'\rangle}$$

$$\text{cpl}_3: \frac{}{\langle res(., a, b), \Phi''\rangle \Rightarrow \langle ., \Phi'\rangle}$$

Figure 4.6: Composition and complementary rules for μDSL

We can unfold the meaning of the expression $tsk(9, 3)$ *by*

$$\psi \wedge \Diamond_{=9}\ \psi \wedge \Diamond_{=18}\ \psi \wedge \Diamond_{=b}\ \psi, \tag{4.6}$$

which is as big as the required bound, which in this case is $b = 27$.

For the second alternative making use of the always operator, we define it by the formula

$$\mathsf{RE}_{(\tau_1)} \wedge \Box_{<b}\ \mathsf{RE}_{(\tau_1)} \rightarrow \left(\Diamond_{=p}\ \mathsf{RE}_{(\tau_1)}\right) \wedge \psi, \tag{4.7}$$

where b *is the upper bound, equal to* $27 + 9$*, and* $p = 9$ *is the task period, which means the starting point of the execution of a task.*

We decided to adopt the second option for μDSL, since in terms of synthesis the result will be more succinct.

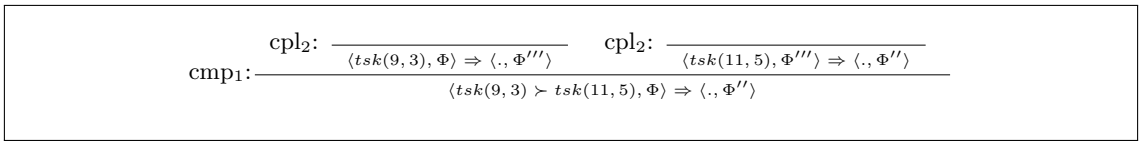


Figure 4.7: Inference tree for the Example 15

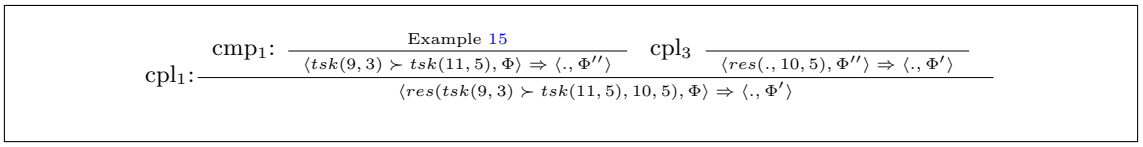


Figure 4.8: Inference tree for the Example 16

Example 15. *Let us begin by a simple example using the expression* $tsk(9,3) \succ tsk(11,5)$, *and identify* $tsk(9,3)$ *by* τ_1 *and* $tsk(11,5)$ *by* τ_2. *Applying the rules* cpl_2 *and* cmp_1, *we can construct the inference tree depicted in the Figure 4.7. We get* Φ'' *equal to*

$$\left(\mathsf{RE}_{(\tau_2)} \rightarrow \left(\left(\Diamond_{=11}\ \mathsf{RE}_{(\tau_2)}\right) \ominus \left(\mathsf{ST}_{(\tau_2)} \vee \mathsf{RS}_{(\tau_2)} \vee \mathsf{SL}_{(\tau_2)} \vee Fl(\phi'')\ \mathrm{U}_{<11}\ \mathsf{SO}_{(\tau_2)}\right)\right)\right) \wedge \left(\mathsf{RE}_{(\tau_2)} \rightarrow \int^{11} \mathsf{ST}_{(\tau_2)} \vee \mathsf{SL}_{(\tau_2)} \vee \mathsf{RS}_{(\tau_2)} \vee \mathsf{SO}_{(\tau_2)} = 5\right) \wedge \Phi''',$$

where Φ''' *is equal to*

$$\left(\mathsf{RE}_{(\tau_1)} \rightarrow \left(\left(\Diamond_{=9}\ \mathsf{RE}_{(\tau_1)}\right) \ominus \left(\mathsf{ST}_{(\tau_1)} \vee \mathsf{RS}_{(\tau_1)} \vee \mathsf{SL}_{(\tau_1)}\ U_{<9}\ \mathsf{SO}_{(\tau_1)}\right)' label1\right)\right)' unt1 \wedge \left(\mathsf{RE}_{(\tau_1)} \rightarrow \int^{9} \mathsf{ST}_{(\tau_1)} \vee \mathsf{SL}_{(\tau_1)} \vee \mathsf{RS}_{(\tau_1)} \vee \mathsf{SO}_{(\tau_1)} = 3\right)' dur1 \wedge \Phi,$$

and the filter function $Fl(\Phi''')$ *returns the formula*

$$\mathsf{RE}_{(\tau_1)} \vee \mathsf{ST}_{(\tau_1)} \vee \mathsf{RS}_{(\tau_1)} \vee \mathsf{SL}_{(\tau_1)} \vee \mathsf{SO}_{(\tau_1)}.$$

Note that the filter Fl makes use of labels. For the next example, let us denote Φ'' *by* Ψ_1.

Example 16. *Let us assume the expression* $res(u', 10, 4)$, *where* u' *is equal to* $tsk(9,3) \succ tsk(11,5)$ *as in the Example 15. Applying the rules* cpl_1, cmp_1, cpl_2, *and* cpl_3, *we get the inference tree depicted in Figure 4.8. We get* Φ' *equal to*

$$\left(\Diamond_{=10}\ \mathsf{RN}\right) \wedge \left(\mathsf{RN} \rightarrow \int^{10} Fl(\Phi'') < 4\right).$$

Note that in this case $\Phi' \wedge \Psi_1$ *is false, since* Φ' *conflicts with* Ψ_1 *due to the execution time of the tasks that exceed 4 time units. Let us now denote* $\Phi' \wedge \Psi_1$ *by* Ψ_2 *for simplicity.*

Finally, we get the final formula

$$\left(\mathsf{RN} \ominus \left(\mathsf{RE}_{(\tau_1)} \ominus \mathsf{RE}_{(\tau_2)}\right)\right) \wedge \Box_{<b}\ \Psi_2,$$

where b is the least common multiple of the expression.

4.5 Timing guarantees by hierarchy of monitors

Timing correctness regarding the execution of explicit time monitors. Knowledge of the length of the traces is required before execution, and for that we define a *bound* over temporal formulas, allowing us to determine a map from time to event size. The calculation of temporal bounds for formulas of RMTL-$\int_3$ is then achieved by a recursive algorithm that traverses the inductive structure of the formulas by summing the time window required for each formula. We now give two examples of the calculation of an upper bound for a given formula, and the construction of a flow graph for a given time window.

Example 17. *Let us consider a trace and the formula* $a \ \mathrm{U}_{<10} \ (b \ \mathrm{U}_{<10} \ c)$*, containing propositions* a, b, c *evaluated at time* $t = 0$*. Based on the semantics of temporal operators we achieve the timing bounds* $t \in \,]0, 10[$ *and* $t \in \,]0, 20[$*, for the inner and outer until operators. These time bounds are intervals where the truth values resulting from the evaluation of formulas may change. By the semantic nature of temporal operators, we know that for any* $t \notin \,]0, 10[\cup]0, 20[$ *the truth value is maintained constant, which gives us the desired bound for changes of the evaluation value.*

Example 18. *In order to estimate the amount of time required from the system under observation to couple monitors in a safe manner, we can use a pessimistic approach based on the assumption of a maximum inter-arrival time of events in the system, or we can pre-compute the flow graph of the application. Based on these, we are able to infer how many events will be triggered in a certain time interval. To exemplify the specific case of the latter, we define a time window given by a certain formula using the previous approach. Then, we create a flow graph of the entire system and fix the starting point of the system as depicted in the partial flow pattern of the events under monitoring in the Figure 4.3. From label* $\mathsf{ST}_{\omega_1,\tau_1}$ *to* $\mathsf{SO}_{\omega_1,\tau_1}$*, where* ST *corresponds to the beginning of the execution and* SO *corresponds to the end of the execution, we have the flow of the main task composed by three paths , and from label* $\mathsf{ST}_{\omega_2,\tau_2}$ *to* $\mathsf{SO}_{\omega_2,\tau_2}$*, we have the optional task, which includes* $\mathsf{EV}_{(1)}$ *and* $\mathsf{EV}_{(2)}$*. In summary, we have at most four events between* ST *and* SO *and the optional task two events. The figure also depicts the dependencies of events, and allows us to estimate the required relative time for some events.*

Altogether, these examples combine temporal settings of the monitors and the system itself: the first one give us the amount of time that we need to wait for a verdict (minimum time granularity); the second one helps us to find the period for a monitor based on the time behavior of the system under monitoring as well as to estimate the WCET of the monitor (i.e, the time complexity times a constant).

Timing guarantees of the hard real-time systems are commonly pessimistic [Shin and Lee, 2003]. Given that, it is not good to have monitors always executing in constant time since they may consume more time than required in average. In order to produce coherent timing verdicts of monitors without assuming any specific scheduler, a hierarchy of monitors should be employed. The main monitor requires to execute in constant time to supervise the other monitors that can be executing without any restriction of time. Given that, as the time elapses the main monitor is ensuring the timing guarantees of the other monitors and then these monitors are supervising the main application. Now, we are able to use our framework to settle on any real-time scheduler.

The idea behind a hierarchy of supervising monitors is to obtain a monitor that is correct-by-construction and executes in constant-time and constant-space. This allows us for adaptability of new monitors, as well as to incorporate new system functions. In order to give constant-time implementation of a monitor, we need to fix the sample size for the trace that the supervisor monitor uses to incrementally evaluate, and use the symbol-based execution for arbitrary n steps. However, we do not have guarantees that the maximum delay detection will be ensured. For that we need to consider the rate of the events that scheduler and monitors trigger. It is relatively simple since monitors are time/event triggered or both. Since counting events is constant time, we have a monitor that will count the events in order to verify if they are greater than the amount of events allowed by the system. Note that this is safe by itself since the assumption is also monitored.

Note that none of the related works have focus on an hierarchy of trusted monitors. At most, they assume that the monitors execute as fast as possible and when there is no *real-time operating system* (RTOS), the scheduling is employed by the hardware interrupt routines [Pike et al., 2010].

Summary

In this chapter, we have presented the formalization of periodic resource models extended with dependent tasks. Based on that we have constructed the analysis for the presented framework in order to discharge properties statically by means of an offline analysis, and at execution time employing runtime monitors. For constructing the skeleton of the monitoring sketch, we have introduced the μDSL language, which we believe has the potential to become an important artifact for the real-time community, embedding the same language as the one we have introduced in Chapter 3 to synthesize monitors.

This is the novelty of our approach. Instead of being too generic, it allows us to define more concrete/specific constraints about the execution of the system under observation, and at

the same time specifying runtime monitors. For the cases where there are less constraints, the output of the offline analysis will be successful as well, including the extension for multi-core systems where cores and memory regions are automatically assigned. Moreover, proofs are generated for each sketch giving us a great confidence over the analysis just by assuming the synthesis mechanisms. The practicability of our approach depends on both synthesis steps, which are of major importance.

In terms of the practical implementation of the framework proposed and described in the chapter, we follow an approach that consists in: 1) synthesis mechanism for functional language (and then extended to imperative languages such as C++); 2) synthesis mechanism for SMT solvers such as Z3 , and 3) the framework including a proper language and tools to combine both offline and online mechanisms. When mixing these techniques we are able to carry out safe RV of hard real-time systems.

Chapter 5

Evaluation

Over the past decades several approaches for schedulability analysis have been proposed for both uni-processor and multi-processor real-time systems [Davis and Burns, 2011]. Although different techniques are employed, very little has been put forward in using formal specifications, with the consequent possibility for mis-interpretations or ambiguities in the problem statement [Cerqueira et al., 2016].

Moreover, the major effort in the research community working on controller design for real-time embedded systems is the design of *physical models* rather than *model synthesis* techniques and associated formal verification approaches [Ranjbaran and Khorasani, 2010]. Even when formal synthesis and verification methods are used, the techniques for enforcing time isolation are generally discarded and delegated to the capabilities of non-formally/partially verified RTOSs [Andronick et al., 2016, Meier et al., 2015].

In this chapter, we describe the application of the techniques and the framework presented in the preceding chapters, and evaluate their usability regarding the safe inclusion of monitors in a working environment as well as the monitor synthesis from RMTL-$\int_3$ language. We will begin by describing the usefulness of our approach in the context of offline schedulability analysis, and later on showing evidence of the effectiveness for schedulability analysis of uni- and multi-processor systems without runtime monitors. Then, we introduce the case study for RV of lightweight avionic systems making use of the RV-RMTL-$\int$ framework for monitoring control systems. Finally, we discuss the kind of properties we are able to deal with, as well as the results achieved in verifying them.

5.1 Application of μDSL for offline schedulability analysis

Along almost forty years, a bewildering diversity of schedulability tests for hard real-time systems has been proposed to address the constrains imposed by the required timing predictability. These tests vary considerably in their complexity, expressivity, and target scheduling policies (e.g., fixed task or job priority, preemptive or non-preemptive). The literature [Audsley et al., 1995, Fidge, 1998] reveals that generally schedulability testing works by assuming a worst-case scenario and checking that each of the involved tasks gets a sufficient allocation of shared resources or jobs complete before their deadlines. Although in multi-core the same does not naturally happen, cases that are not "the worst" will also succeed.

The reasons for adopting a logic-based paradigm for schedulability analysis are: it becomes more comprehensive and expressive; it rules out potential specification incoherences typical of informal specifications; and it has some benefits relatively to the available analysis, not in terms of efficiency but in terms of being easily extendable for monitoring approaches such as the acquisition of the maximum detection delay of a task as in [Zhu et al., 2009]. As further context on offline scheduling using temporal logic, we note that:

1. the outcome of a classical schedulability analysis is typically a verdict for a certain set of tasks, but no counter-examples are shown if the set of tasks is not schedulable;
2. the behavior of the scheduler is *assumed* rather than being explicitly included in the schedulability test;
3. the timing description of the tasks is the unique data provided by classical analysis methods (*i.e.*, offsets, jitters, periods, deadlines);
4. standard approaches are not possible to extend with other useful properties such as monitoring and enforcement of real-time properties [Pinisetty et al., 2013, Pike et al., 2010], due to the restricted definition of their sets of tasks (*e.g.*, defining a bound for two consecutive instructions, the inter-arrival time of an event);
5. some real-time systems literature [Zhu et al., 2010, 2009] commonly considers the estimation of an arrival rate, which implies minimization and produces significant issues (e.g., under and over estimations, local minimums and maximums, *etc.*).

This work integrates the description of the scheduling behavior with the schedulability analysis, which enables the generation of counter-examples when the system is not schedulable. These counter-examples are fundamental for the system designer to understand and adapt the design accordingly. Although giving an unsatisfiable answer is, in general, faster,

it is not straightforward to draw a readable counterexample as the SMT solver normally relies on getting the minimal unsatisfiable core.

The present schedulability analysis consists in the evaluation of a formula over a trace (or a set of traces) produced by a periodic resource model where tasks execute along a fixed priority scheduling. In order to decrease the state space search we might assume for uni-core scheduling the critical instant theorem [Liu and Layland, 1973]. This *assumption* would reduce our problem to just one trace acceptance for a set of logic properties and would allow us to identify the relevant traces and combine our approach with the foundational real-time systems theory. However, this does not work for multi-core scheduling and is thus not sufficiently generic for our purposes.

Our schedulability decision problem is indeed a satisfiability problem over a trace regarding a RMTL-$\int$ formula. The general schedulability problem for tasks/resources is described in the following definitions.

Definition 22. Let $\{\tau_1, \tau_1, \ldots, \tau_n\} \subseteq \mathcal{T} \subseteq \Gamma$ be a set of tasks with arbitrary size n. The set of tasks are schedulable according to a fixed priority if and only if there exists an event sequence such that $\mathsf{PRM}(\omega_1)$ holds for some ω_1 equal to $(\mathcal{T}, l, l, fp)$ with l a sufficient large number, and fp the fixed priority policy.

Definition 23. Let $\{\omega_1, \omega_2, \ldots, \omega_m\} \subseteq \Omega$ be resource models with arbitrary size m. The resource models are said to be *schedulable* if and only if, there exists an event sequence such that $\mathsf{PRM}(\omega_1) \wedge \mathsf{PRM}(\omega_2) \wedge \cdots \wedge \mathsf{PRM}(\omega_m)$ is satisfied, and the duration of the found event sequence is greater than or equal to hyper period among resources.

Informally, these definitions lead us to state that there exists in the past sufficient resources to meet the deadlines of all tasks in the periodic resource model if this resource model acts as specified (i.e., behaves accordingly).

Our schedulability analysis for several period resource models relaxes the truth notion of the WCET. This means that the WCET of a task (or set of tasks) can be erroneously estimated, and ensures that the remaining resource models are also schedulable, which is a property of great interest for multi-core scheduling where anomalies can happen.

Next, we will consider a simple fixed priority schedulability test with implicit deadlines, and then move forward to a more elaborated example based on multi-core scheduling. For both we will use μDSL (introduced in Chapter 4 as part of the RV-RMTL-$\int$ framework) to encode simple expressions, since it is more succinct.

$$prop_{fm} \triangleq \mathsf{RU}(c_0,\tau_1)\vee\mathsf{SO}(c_0,\tau_1)\vee\mathsf{RU}(c_0,\tau_2)\vee\mathsf{SO}(c_0,\tau_2)\vee\mathsf{RU}(c_0,\tau_3)\vee\mathsf{SO}(c_0,\tau_3)$$

$$init \triangleq \mathsf{RN}(c_0)\ \mathrm{U}_{<2}\ (\mathsf{RE}(c_0,\tau_1)\ \mathrm{U}_{<2}\ (\mathsf{RE}(c_0,\tau_2)\ \mathrm{U}_{<2}\ \mathsf{RE}(c_0,\tau_3)))$$

$$\Box_{<60}\ \mathsf{RN}(c_0)\rightarrow\left(\Diamond_{=60}\ \mathsf{RN}(\omega)\right)\wedge\textstyle\int^{60} prop_{fm}<50$$

$$\Box_{<60}\ \mathsf{RE}(c_0,\tau_1)\rightarrow\left(\Diamond_{=20}\ \mathsf{RE}(c_0,\tau_1)\right)\wedge\left(\mathsf{RE}(c_0,\tau_1)\ \mathrm{U}_{<2}\ \left(\mathsf{RU}(c_0,\tau_1)\vee\mathsf{RU}(c_0,\tau_3)\vee\mathsf{SO}(c_0,\tau_3)\ \mathrm{U}_{\leq 20}\ \mathsf{SO}(c_0,\tau_1)\right)\right)$$

$$\Box_{<60}\ \mathsf{RE}(c_0,\tau_2)\rightarrow\left(\Diamond_{=15}\ \mathsf{RE}(c_0,\tau_2)\right)\wedge\left(\mathsf{RE}(c_0,\tau_2)\ \mathrm{U}_{<2}\ \left(\mathsf{RU}(c_0,\tau_2)\vee\mathsf{RU}(c_0,\tau_1)\vee\mathsf{SO}(c_0,\tau_1)\vee\mathsf{RU}(c_0,\tau_3)\vee\mathsf{SO}(c_0,\tau_3)\ \mathrm{U}_{\leq 15}\ \mathsf{SO}(c_0,\tau_2)\right)\right)$$

$$\Box_{<60}\ \mathsf{RE}(c_0,\tau_3)\rightarrow\left(\Diamond_{=10}\ \mathsf{RE}(c_0,\tau_3)\right)\wedge\left(\mathsf{RE}(c_0,\tau_3)\ \mathrm{U}_{<2}\ \left(\mathsf{RU}(c_0,\tau_3)\vee\mathsf{RU}(c_0,\tau_2)\vee\mathsf{RU}(c_0,\tau_1)\vee\mathsf{SO}(c_0,\tau_1)\vee\mathsf{SO}(c_0,\tau_2)\ \mathrm{U}_{\leq 10}\ \mathsf{SO}(c_0,\tau_3)\right)\right)$$

$$\Box_{<60}\ \mathsf{RE}(c_0,\tau_1)\rightarrow\textstyle\int^{20}\mathsf{RU}(c_0,\tau_1)\vee\mathsf{SO}(c_0,\tau_1)=9$$

$$\Box_{<60}\ \mathsf{RE}(c_0,\tau_2)\rightarrow\textstyle\int^{15}\mathsf{RU}(c_0,\tau_2)\vee\mathsf{SO}(c_0,\tau_2)=8$$

$$\Box_{<60}\ \mathsf{RE}(c_0,\tau_3)\rightarrow\textstyle\int^{10}\mathsf{RU}(c_0,\tau_3)\vee\mathsf{SO}(c_0,\tau_3)=3$$

$$init$$

Table 5.1: Expansion of the $\mathsf{PRM}(c_0)$ where c_0 means $core_0$

5.1.1 Two settings for schedulability analysis

μDSL in uni-core setting. To demonstrate the effectiveness of the schedulability analysis using μDSL, we introduce a synthetic workload. Consider as example the workload composed by one component $(60, 50)$, which executes at each hyper period three tasks with parameter pairs $(20, 9)$, $(15, 8)$ and $(10, 3)$, with available 50/60 time units for executing. The first element of the tuple is the period and the second the deadline/budget. In μDSL, the expression describing the example is

$$server_0\left[\left(tsk_{ts1}^{(20,9)} \succ tsk_{ts2}^{(15,8)}\right) \bowtie tsk_{ts3}^{(10,3)}\right]_{(60,50)}, \tag{5.1}$$

which specifies that $ts1$ has higher priority than $ts2$, and $ts3$ executes arbitrarily with $ts1$ and $ts2$.

Usage of events as specified in the RV-RMTL-$\int$ framework is more adequate for runtime monitoring purposes. Due to the overhead that resume and sleep events may cause when using SMT solvers and the ability to infer when a task sleeps/stops occurs based on non consecutive events, we will adopt only three events per task, RE, RU (meaning ST, RS or SL) and SO. Based on that, we have automatically formulated the set of formulas described in Table 5.1 from Expression 5.1 using the proposed synthesis algorithm for SMT solvers. The same table also includes a trace that satisfies the given specification. Note also that other events can be further considered as required. The reader is referred to Appendix B for a more detailed example of a complete synthesis.

```
(define-fun indicator ((mt Time)) Int
  (ite (= (compute_p trace mt p_a) TVTRUE) 1 0)
)

(declare-fun evaln ((Time)) Int)
(assert (= 0 (evaln 0)))
(assert (forall ((x Int)) (=> (> x 0) (= (evaln x ) (+ (evaln (- x 1))
   (indicator x) )))))

(assert (< (evaln 10) 9 ))
```

Listing 5.1: Example of a RMTL-$\int$ duration term encoding using SMT-Libv2

μDSL in multi-core setting. A specification for a multi-core setting, making use of the previous expression, can be expressed as

$$server_0 \left[\left(tsk_{ts1}^{(10,8)} \succ tsk_{ts2}^{(20,5)} \right) \bowtie tsk_{ts3}^{(27,7)} \right]_{(1,1)} \overset{c}{\mapsto} core_0 \parallel$$
$$server_1 \left[tsk_{ts4}^{(33,4)} \succ tsk_{ts5}^{(6,2)} \right]_{(1,1)} \overset{c}{\mapsto} core_1, \tag{5.2}$$

where instead of specifying the amount of execution time allowed for each resource the expression assigns for $server_0$ and $server_1$ the pair $(1, 1)$. This means that all available resources in the $server_0$ are executing in isolation in the $core_0$ as well as the resources of $server_1$ in $core_1$.

For both settings, the next step of the approach (introduced in Chapter 4) consists in the transformation of a specification written in μDSL into an equivalent RMTL-$\int$ specification. We can then check the satisfiability of a scheduling property over the generated set of formulas like for instance checking if task $ts1$ can execute more than 9 time units. Next, we convert this formula into the SMT-LIBv2 [Barrett et al., 2015] language using our tool (described in Appendix B) and delegate the reasoning to the Z3 solver [de Moura and Bjørner, 2008].

To better exemplify how the process is done, let us consider the Listing 5.1 that shows an incomplete candidate encoding of the interval-based semantics for the RMTL-$\int$ duration term. The uninterpreted function $\texttt{compute}_p$ evaluates a proposition at the instant `mt`, and $\texttt{p}_a$ is a proposition representing an event. It is true from the beginning of the event's occurrence until the next event is triggered in the system. Our goal is to find a trace (or set of traces) that satisfies these constraints, henceforth if the answer we obtain is *unsat* then the system cannot be scheduled (the constraints are somehow inconsistent); otherwise, we have a flow of the system for which these constraints result in a schedulable behaviour.

ID	Formula	Checked	Performance
(a)	$p \wedge \Box_{<b_1}\ p \rightarrow \Diamond_{=2}\ p$	✓	
(b)	$(p \vee q)\ \mathrm{U}_{<b_1}\ r$	✓	
(c)	$\int^{b_1} p < 3$	✓	
(d)	$((p \vee q)\ \mathrm{U}_{<b_1}\ r) \wedge \int^{9} r < 2$	✓	
(e)	$((p \vee q)\ \mathrm{U}_{<b_1}\ r) \wedge 10 < \int^{9} r$	unsat	
(f)	$\Diamond_{<b_1} p \wedge \Box_{<b_2} \neg p$	unsat	
(g)	$\Box_{<b_2}(a \vee b)\ \mathrm{U}_{<b_1}\ r$	✓	

Table 5.2: Heat maps for performance comparison using the rmtld3synth tool for synthesization and the Z3 solver for checking satisfiability

Comparatively to classic approaches, it is clear that this type of reasoning allows us to construct and extend our constraints easily, without the need to reformulate every step of the analysis (it is a constructive approach). Note also that the expressiveness to deal with temporal order is of extreme importance when dealing with systems depending on time, which sets of inequalities and equalities alone cannot provide. It is therefore important to reuse such sets of (in-)equalities and combine them with logic connectives to get a fine-grained description of the system. Furthermore, the recent developments of SMT solvers positively impact our approach, namely due to the efficiency of the underlying reasoning methods that increase the chances of constructing the proofs we need in a fully automatic way.

5.1.2 Experimental results

The setup employed in our experimental evaluation was based on an Intel Core i3-3110M at 2.40GHz CPU with 8 GB of RAM memory, and running Windows 10 Embedded x86 in a virtual machine running on a Fedora 23 X86'64 host.

For RMTL-$\int_3$ formulas. Currently, it is not possible to devise a fair evaluation comparison for our approach since there are no available tools that consider duration terms in the way we consider in this work. In order to provide some insight about the feasibility of our technique, we have measured the times taken by the Z3 SMT solver to prove satisfiability of a set of specifications, as shown in Table 5.2. We have considered different structures for the presented formulae. The goal is to show indicators of the feasibility of the approach on sets of formulae with heterogeneous structural schemes, as we would expect to occur in a real-life example.

The time required to solve formulas is not directly related with a formula's complexity or length, as formula (a) indicates when compared to (c). Note that formulas containing durations are slower in average to solve than formulas containing only temporal operators, as confirmed by the time it took to solve the satisfiability of formula (b) when compared to formula (c). Furthermore, a mix of both temporal operators and durations does not mean slower times as exhibited in the case of formula (d). We also note that showing that a formula is unsatisfiable is in general faster than proving satisfiability. The formula (e) from Table 5.2 is an example of this phenomenon. Finally, formula (g) show that nested temporal operators could grow exponentially. Note that b_1 and b_2 are sampled at increments of 5 from 5 to 50, ■ $<$ 1s and ■ $=$ 100s, and black cells mean a timeout (more than 150s).

More complex examples can be seen in the tool's repository [De Matos Pedro, 2018]. Our experimental results indicate that this method can indeed be feasible for small sets of tasks and resource models.

For μDSL expressions. Experiments using μDSL are described in Table 5.3. The results indicate that this approach does not scale. However, it is very impressive that it was possible to obtain in a few hours results for such highly nested formulas as shown in the table. Note also that |U| means the number of until operators in the formula, and $|\int|$ the number of duration terms. The experiments also show that the results are not dependent of the number of constraints, but on the size of the required input sequence. As the case of $core_0\left[tsk_{ts1}^{(9,8)} \succ tsk_{ts2}^{(3,1)}\right]_{(20,8)}$ getting an unsatisfiable result is faster than getting a satisfiable result when using only one task (i.e, the formula $core_0\left[tsk_{ts1}^{(5,2)}\right]_{(10,10)}$). We use the operators $<$, $>$ to give an upper and lower bound to the time that we require to satisfy the formula.

5.2 Lightweight Autopilot Systems: the case study

In fact, the most common models in the market – excluding the military-grade ones – are not required to follow the rigorous software development processes that are used in commercial avionic systems, mostly because they are small, cheap, and appear to be inoffensive. Furthermore, multi-copters do not have any special inherent stability mechanism, and are very dependent on their control software [Müller and D'Andrea, 2014]. Paradoxically, they are simpler than helicopters but also unsafer, since the latter provide auto-rotation maneuvers that allow them to glide to the ground and still land vertically [Hoffmann et al., 2007].

ID	Expression	|U|	$\lvert \int \rvert$	t(s)	sat
(a)	$server_0\left[tsk_{ts1}^{(5,2)}\right]_{(10,10)}$	5	3	13.55	✓
(b)	$server_0\left[tsk_{ts1}^{(9,6)} \succ tsk_{ts2}^{(3,1)}\right]_{(9,8)}$	10	5	3.05	✓
(c)	$server_0\left[tsk_{ts1}^{(10,2)} \succ tsk_{ts2}^{(10,2)} \succ tsk_{ts3}^{(10,3)}\right]_{(10,10)}$	13	7	< 10800	✓
(d)	$server_0\left[\left(tsk_{ts1}^{(20,9)} \succ tsk_{ts2}^{(15,8)}\right) \bowtie tsk_{ts3}^{(10,3)}\right]_{(60,50)}$	13	7	timeout	✗
(e)	$srv_0\left[\left(tsk_{ts1}^{(10,2)} \succ tsk_{ts2}^{(5,1)}\right) \bowtie tsk_{ts3}^{(5,2)}\right]_{(10,8)} \parallel srv_1\left[tsk_{ts4}^{(5,1)}\right]_{(10,5)}$	18	10	< 16800	✓
(f)	$server_0\left[\left(tsk_{ts1}^{(10,2)} \succ tsk_{ts2}^{(5,1)}\right) \bowtie tsk_{ts3}^{(5,2)}\right]_{(1,1)} \overset{c}{\mapsto} core_0 \parallel server_1\left[tsk_{ts4}^{(9,6)} \succ tsk_{ts5}^{(3,1)}\right]_{(1,1)} \overset{c}{\mapsto} core_1$	20	10	< 14400	✓
(g)	$server_0\left[tsk_{ts1}^{(9,8)} \succ tsk_{ts2}^{(3,1)}\right]_{(20,12)} \triangleleft \left(\mathsf{RU}_{(server_0,ts1)} \twoheadrightarrow \mathsf{RU}_{(server_0,ts2)}\right)$	12	5	< 11000	✓

Table 5.3: μDSL experimental results

We will now show an example that illustrates the usage of an autopilot instrumented with runtime monitors capable to observe the execution of multiple resource models in order to increase the timing confidence of the autopilot's control loop. Our approach uses an offline algorithm for formula simplification, and an online evaluation procedure that can be directly applied for the synthesis of runtime monitors. We will begin by presenting an example of application of Algorithm 1 (already introduced in Chapter 3) for monitoring the budget of a set of *Resource models*s (RMs); then we will present the empirical validation of the complexity results for Algorithm 2 (also presented in Chapter 3). In the remaining part of this chapter, we will introduce two use cases followed by the strong evidence of the feasibility of the runtime monitoring approach.

Let us now recall the concept of resource model (RM). RMs are servers capable to ensure timed resource isolation between tasks. If they are constrained periodically, we define them using a replenishment period and a budged supply. The budget supply is available as time elapses, and is renewed at each period by the resource model. Elastic periodic RMs are resource models containing *elastic coefficients* (similar to spring coefficients in physics) to describe how a task can be compressed when the system is overloaded, and manage imprecise computation. Naturally, the coefficients need to be constrained (linearly or non-linearly) before execution. Intuitively, the idea is to check the coefficients according to the polynomial constraints using our static phase, and provide the simplified formulas for the further runtime evaluation phase.

Let us now extend Example 4 for multiple RMs, considering without loss of generality the case of two RMs. We will use indexed formulas ϕ_{m_i}, ψ_{m_i} with $0 \leq i < n$, $n = 2$, and let

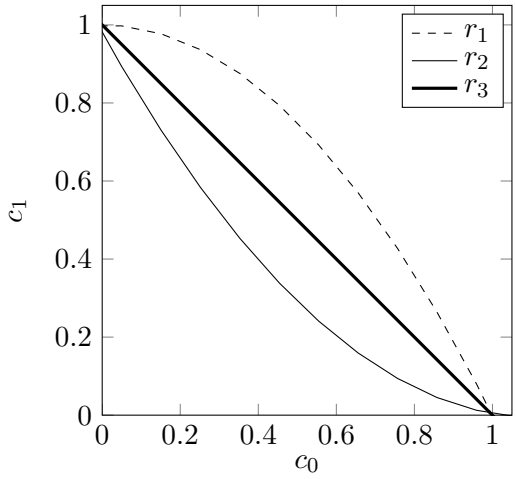


Figure 5.1: Linear, concave and convex restriction for c_0 and c_1

α_i, β_i be indexed constants. For measuring the budgets of two resource models we could use the following invariant:

$$\bigwedge_{i=0}^{n-1} \left(\Box_{<v}\ \phi_{m_i} \rightarrow \left(0 \leq \sum_{j=0}^{n-1} x_j < \beta_i \wedge x_i = c_j \times \int^{\alpha_i} \psi_{m_i} \right) \right) \wedge r_m$$

where v is arbitrarily large, c_i is a coefficient indexed at i that mean different weights for each RM (two in this setting), and r_m is a constraint formula over the free variables c_0 and c_1.

The problem is then to find values for c_0, c_1 satisfying the constraints

$$r_1 := \frac{1}{250}\left(245 - 444 \times c_1 + 200 \times c_1 \times c_1\right) = c_1,$$
$$r_2 := 1 - c_0 = c_1, \text{ or}$$
$$r_3 := 1 - c_0 \times c_0 = c_1,$$

as shown in Figure 5.1, based on two duration observations over the formulas ψ_{m_0} and ψ_{m_1}. Note that r_m is replaced by one of these constraints, namely r_2, and $0 \leq c_0$, $0 \leq c_1$ holds. r_1 and r_3 are only exemplifications of other possible constraints.

We will use Algorithm 1 for discarding possible conflicts, and decompose the formulas into sub-formulas that are free of quantifiers. Let us simplify the previously defined invariant for two resource models where the coefficients c_0 and c_1 are existentially quantified and constrained by r_2. After some transformations on the formula and assuming that both resource models have the same settings (i.e., β_0 is equal to β_1 and α_0 is equal to α_1), we obtain

$$\phi^1_{\not<} := \Box_{<v} \left(\left(\phi_{m_0} \rightarrow \phi^2_{\not<} \right) \wedge \left(\phi_{m_1} \rightarrow \phi^2_{\not<} \right) \right),$$

such that

$$\phi^2_{\not<} := a = \int^{\alpha_0} \psi_{m_0} \wedge b = \int^{\alpha_1} \psi_{m_1},$$

and

$$\phi^1_< := \exists c_0\ c_1\ .\ 0 \leq c_0 \times a + c_1 \times b < \beta_0 \wedge r_3$$

holds. The duration terms $\int^{\alpha_0} \psi_{m_0}$ and $\int^{\alpha_1} \psi_{m_1}$ have been replaced by the logic variables a and b, and the free logic variables x_0 and x_1 have been erased since the duration terms evaluate at the same time. We will then have an isolated formula, and apply CAD to determine if $\phi^1_<$ is satisfied. If it is, then we directly replace $\phi^2_{\not<}$ by *true*, otherwise we have the bounds that satisfy $\phi^2_{\not<}$. For this case, we obtain for $\phi^1_<$ the decomposition

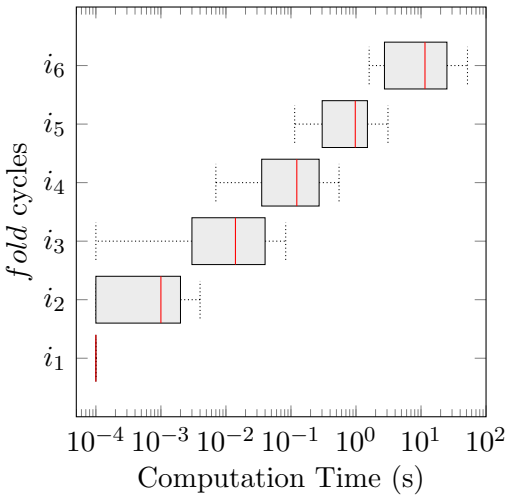
$$(a < 0 \wedge b \geq 0) \vee 0 \leq a < 10 \vee (a \geq 10 \wedge b < 10).$$

Intuitively, we may think on the instances $c_0 = 0$ and $c_1 = 1$, and $c_0 = 1$ and $c_1 = 0$. After this step, the simplified bounds are ready to be evaluated by the online method. Note that we cannot proceed with the monitoring step without removing all the free variables since our monitoring algorithm does not support solving inequalities at runtime. We also have to justify that the usage of runtime solvers is difficult to apply on real-time embedded systems since the demand of computation resources is in the majority of the cases unavailable.

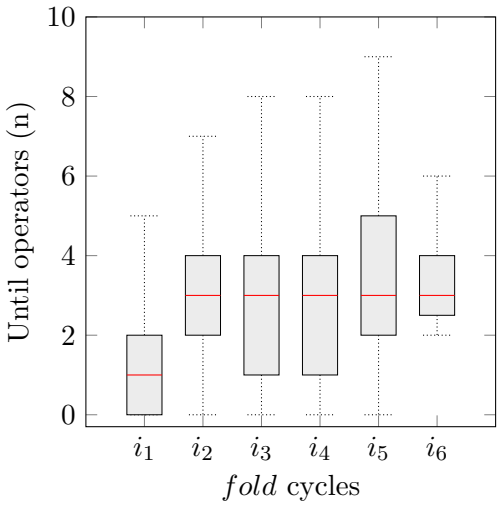
Let us now discuss the complexity of Algorithm 2 and establish an empirical comparison with the bounds presented in the Chapter 3. We observe that the generation of nested durations is more critical on average than the nesting of temporal operators. This result matches the semantics of both terms and formulas, since the duration terms can integrate any indicative function provided for any trace, unlike the until operator that requires a successful trace to maximize its search. Consider Figure 5.2c, where the boxes i_1 to i_6 are respectively the intervals $]10^j, 10^{j+1}]$ for all $j \in [1, 7[$. They represent the number of cycles performed by folding functions. The results confirm that as the number of until operators stabilizes and the number of duration operators increases, the computation time also increases at a higher rate due to the presence of durations. This occurs for *generated* uniform formulas and traces; deep nesting of until operators and nested durations is unlikely to occur in hand-written specifications (it has not been clearly confirmed whether they are useful for real-life applications). The experiments confirm the theoretical complexity bounds obtained earlier (Figure 5.2d). We have performed the experiments on an Intel Core i3-3110M at 2.40GHz CPU, and 8 GB RAM running Fedora 21 X86'64.

5.2.1 Use cases with RMTL-$\int_3$.

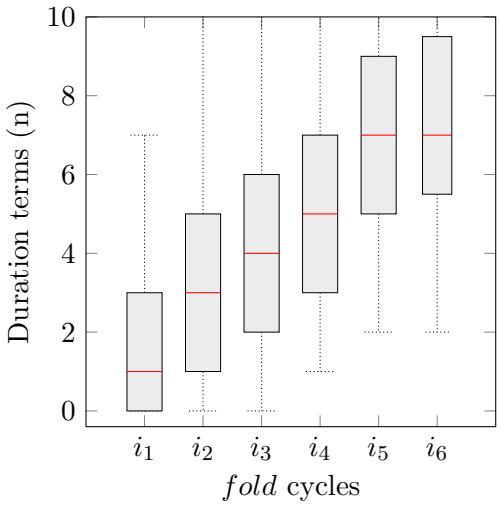
The adopted formalism supports an explicit notion of time that is required for the timing analysis of RTSs. Support of inequalities, durations and quantification over these, increases the expressiveness of classic temporal logics to specify explicit timing settings, filling a gap



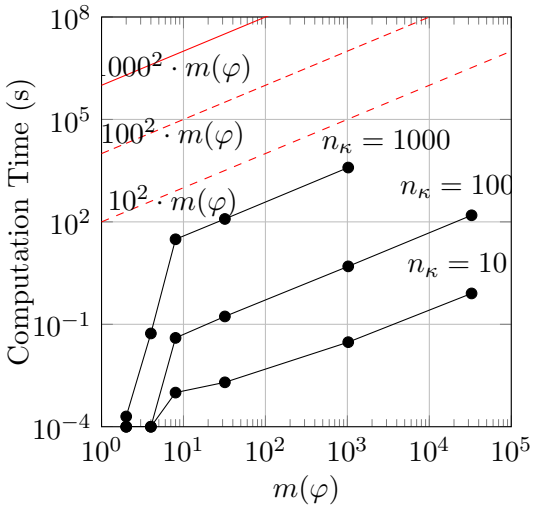
(a) execution cycles of fold functions vs. computation time, $m(\varphi) = 2^5 - 1$ and $n_\kappa = 1000$



(b) number of until operators vs. execution cycles of fold functions, $m(\varphi) = 2^5 - 1$ and $n_\kappa = 1000$



(c) number of duration terms vs. execution cycles of fold functions, $m(\varphi) = 2^5 - 1$ and $n_\kappa = 1000$



(d) computation time vs. formula size constructed with nested Until operators

Figure 5.2: Experimental validation of the complexity results

in the common specification languages for RTSs. Increasing the expressiveness of temporal logics may introduce decidability issues; the interest of decidable fragments, like RMTL-$\int_3$, is that the existence of an effective procedure that always evaluates any formula in any model as a truth value is guaranteed. In practice, the existence of this procedure implies that a monitor always terminates drawing a verdict, which is indeed important in runtime monitoring applications, and even more important in the context of hard real-time systems.

Let a be a coefficient represented by a logic variable. Duration terms of the form $a \times \int^{\eta_1} \psi_1$

can be synthesized if the coefficient a is constrained by polynomial inequalities, or if the coefficient a with distribution *Beta* or *Dirichlet* is employed. Under these restrictions, our tool [De Matos Pedro, 2018] is able to generate monitors that evaluate conditional probabilities of random actions of RTSs. For instance, these monitors can be used to monitor the inflation and the deflation of imprecise tasks, which is required when imprecise computation models are employed. Moreover, the degradation of the system can also be specified by defining liveness properties such as "a task cannot execute for less that 5 time units in one interval of 100 time units".

Two use cases for monitoring of the Ardupilot autopilot framework are described in this section. The first is a simple case that exemplifies the quantification of linearly constrained duration formulas, to illustrate how to generate monitoring conditions in C++. Use Case (2) explores how to encode uncertainty by using polynomial inequalities to constrain quantified duration formulas.

Use Case (1): RM establish amounts of shared resources to be consumed by working tasks in RTSs. Normally, these mechanisms focus on time consumption and ensure *time isolation* between different tasks or sets of tasks. *Periodic RMs* are defined by their *replenishment period* and *budget supply*. Budgets are dynamically available as the time elapses and are replenished at certain defined periods. *Elastic RMs* are an extension of periodic RMs containing *elastic coefficients*, similar to *spring coefficients* in physics. They describe how the execution time of a task can be temporally deflated or inflated by applying n-D geometric region constraints (polynomial inequalities) over resource budgets. These restricted coefficients allow for the system's under-load and over-load to be controlled. Spring coefficients, which are seen as logic variables, define the rate (or constraint) of inflation and deflation of a resource (in our case, processing time) and can be changed during execution. In this use case, these coefficients are governed by linear inequality constraints which dictate the under- and over-loading conditions of a certain set of tasks.

Example 19. *Consider the formula*

$$0 \leq a \times \int^{\pi_1} \psi_1 + b \times \int^{\pi_2} \psi_2 \leq \frac{1}{4}\theta$$

that specifies the resource constraints of two RMs where coefficients are managed according to the linear equation $a = 1 - b$ *for* $a, b \geq \frac{1}{4}$*, that* ψ_1,ψ_2 *are two formulas describing the event releases of two distinct tasks, and that* θ *is the allowed execution time for the RMs. Informally, the formula specifies that both resource models have different budgets when both execute at the same time, which in practice is the case when both RMs interfere in the system. To find the conditions for monitoring we need to quantify the formula, yielding a new formula*

$$\exists_{\{a,b\}} \left(a = 1 - b \wedge a > \frac{1}{4} \wedge b > \frac{1}{4} \wedge 0 \leq a \times \int^{\pi_1} \psi_1 + b \times \int^{\pi_2} \psi_2 \leq \frac{\theta}{4} \right).$$

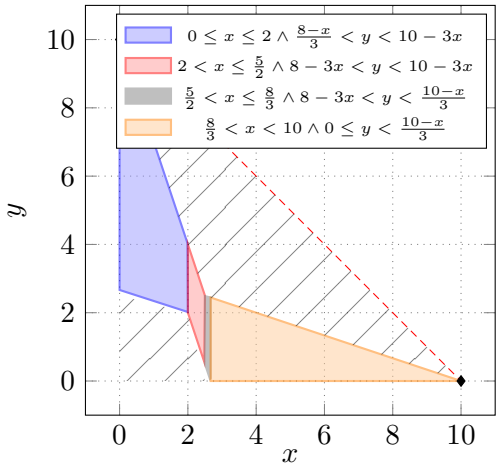


Figure 5.3: Regions of decomposed inequalities with duration x, y and $\theta = 10$

Later, after applying the simplification algorithm described in the Chapter 3, we generate the monitoring conditions from Example 19, as follows:

$$(\textstyle\int^{\pi_1}\psi_1{=}0 \wedge 0 \le \int^{\pi_2}\psi_2 < \theta) \quad \vee \quad \left(0 < \int^{\pi_1}\psi_1 < \frac{\theta}{4} \wedge 0 \le \int^{\pi_2}\psi_2 < \theta - 3\int^{\pi_1}\psi_1\right) \quad \vee$$
$$\left(\textstyle\int^{\pi_1}\psi_1 = \frac{\theta}{4} \wedge 0 \le \int^{\pi_2}\psi_2 \le \frac{\theta}{4}\right) \quad \vee \quad \left(\frac{\theta}{4} < \int^{\pi_1}\psi_1 < \frac{\theta}{3} \wedge 0 \le \int^{\pi_2}\psi_2 < \frac{\theta - \int^{\pi_1}\psi_1}{3}\right) \quad \vee$$
$$\left(\textstyle\int^{\pi_1}\psi_1 = \frac{\theta}{3} \wedge \theta - 3\int^{\pi_1}\psi_1 \le \int^{\pi_2}\psi_2 < \frac{\theta - \int^{\pi_1}\psi_1}{3}\right) \quad \vee \quad \left(\frac{\theta}{3} < \int^{\pi_1}\psi_1 < \theta \wedge 0 \le \int^{\pi_2}\psi_2 < \frac{\theta - \int^{\pi_1}\psi_1}{3}\right),$$

where ψ_1 and ψ_2 are both simplified formulas.

In Figure 5.3 we can see regions where the RMs are able to consume resources or not, as well as regions where they are not able to do so. For instance, the resource B cannot consume any resource if resource A consumes 10 units, and the resource A can only consume more than 4 units if the resource B consumes less than 2 time units, due to resource constraints. For the case of both resources consuming 2.5 units each, the difference between the sum and the execution time indicates that the interference of both resource models executing concurrently is at most 5 time units (it is identified by the hashed region). Intuitively, this constraint means that one resource needs to be deflated when the other resource is inflated and conversely. Note that different regions can be found by modifying the constraints of the scale factor $\frac{1}{4}$, or any of the θ, a or b parameters.

Use Case (2): A conditional probability for a given duration measure for tasks can be specified using this formalism. We will next evaluate the likelihood of the remaining tasks in a system to be unscheduled, based on the overload of a certain task. This example applies in the context of RMs monitoring and also of imprecise computation monitoring. Let a be defined as a coefficient with uncertainty. Any probability distribution that can be described using polynomial inequalities can be encoded using this approach. Here we will focus on the *Beta* distribution only, but other interesting distributions, such as multinomial and Dirichlet distributions, could be equally used.

Let X and Y behave as two random variables with distribution $Beta(a_i, b_i)$ for $i \in 0, 1$. To encode these random variables in RMTL-$\int_3$, we define the *Beta probability density function* (*pdf*) as a constraint of the form

$$\frac{\widehat{f}(1-x, \beta-1)\, \widehat{f}(x, \alpha-1)}{C_\beta},$$

where C_β is simplified and equal to $B(\alpha, \beta)$, and $\widehat{f}$ is the power function. Power functions can be encoded in RMTL-$\int_3$ with the following axiom

$$y = \sqrt[a]{x^b} \;\Leftrightarrow\; x^b = y^a \vee y = x^{\frac{b}{a}},$$

for any $x, y \in \mathbb{R}_{\geq 0}$, $a, b \in \mathbb{Q}_{>0}$. Any function $\widehat{f}$ may now be encoded in RMTL-$\int_3$. The *Beta distribution* $p = f_{\beta,\alpha}(x)$ is now fully defined by

$$y^{a_1} = (1-x)^{b_1} \wedge z^{a_2} = x^{b_2} \wedge \frac{y \times z}{C_\beta} = p,$$

where $a_i, b_i \in \mathbb{N}$, $i \in \{1, 2\}$ are solutions of the formulas $\frac{a_1}{b_1} = \beta - 1$ and $\frac{a_2}{b_2} = \alpha - 1$, and p stands for the probability of the logical variable x in the interval $[0, 1]$.

Intuitively, the idea is to specify non-deterministic actions based on the information provided at execution time. For instance, a system can change its *modus operandis* if for some reason the probability of a given overload is greater than a certain fixed probability threshold. Note that these probabilistic inequality constraints will be used as monitoring conditions. The generation of monitoring conditions based on simplification approaches, as in the Use Case (1), is only required if quantifiers are applied.

Let us consider without loss of generality the case of two tasks, where the first one may have a chance to overload, and the second one should avoid this by self-deflating. The specification of probabilistic coefficients that supports elasticity when overload situations occur is encoded by

$$a = \int^{p_1} \psi_1 \wedge \Box_{<v} \left(f_{\beta,\alpha}(a) < \frac{3}{4} \rightarrow \Diamond_{<p_1+p_2}\, \psi_d \right),$$

where v is arbitrarily large, ψ_d is defined as

$$\int^{p_2} \psi_2 < b \times d,$$

a and b are restricted by one polynomial inequality constraint (e.g., $a = b + 1$), d is the maximum allowed execution time for a task, and ψ_1, ψ_2 are the formulas defined for each of the two tasks (e.g., conjunction of propositions for specifying a certain task or RM). Remark also that p_1 and p_2 are constants which represent the period of the tasks.

5.2.2 Experimental Results

Before discussing the experimental results for the presented use cases, we start by comparing the results presented in Figure 5.2 with the ones presented next, where we show that one element takes in average $400ns$ to be processed using an Intel x86 machine. For that, we re-use the Ocaml source code used to generate the results provided in the Figure 5.2 in order to compare with our present setting.

For comparing both implementations, we have used the following set of RMTL-$\int_3$ formulas: (a) $true\ U_{\leq t}\ \phi$ (eventually); (b) $\phi \rightarrow \Box_{\leq t}\, \psi$ (bounded-invariance); (c) $\Box_{\leq t} \int^t \phi \leq \beta$ (limited-duration); and finally (d) $\phi \rightarrow \int^t \psi \leq \beta$ (bounded-duration).

For each formula we have tested, we have also used different trace sizes ranging from 10 to 10^3. The traces that we consider are selected as the traces that maximize the execution time of each formula evaluation. We have run the experiments on two distinct architectures, namely, the ARM(armv7) and the x86(i686) architectures. The OCaml experiments were only performed on the x86 architecture, while the C++ implementation was tested on both of them.

PixHawk [Meier et al., 2015] board is the target platform to execute periodic monitors that were synthesized from RMTL-$\int_3$ formulas into C++. We also have tested the same implementation using an Intel Core i3-3110M at 2.40GHz CPU with 8 GB of RAM memory, and running Windows 10 Embedded x86 in a virtual machine running on a Fedora 23 X86'64 host.

In the case of the PixHawk board, we have only 256kb of memory RAM for the overall system and we assign at most 90% of the processor usage for these monitoring experiments. From the experimental results presented in Figure 5.4, we can conclude that such monitors execute in polynomial time as the trace increases, which goes according to the theoretical results presented in [De Matos Pedro et al., 2015a]. [1] The stack consumption is also acceptable for PixHawk board. The constant upper dashed line is the maximum stack consumption of 1.76kb for the formula (c), and the other two lines are the lower bounds of the remaining three formulas that have a very similar stack usage. Different lines are depicted in Figure 5.4. They correspond to different execution times and stack experiments: the lines tagged with "ocaml" refer to the execution of the original evaluation algorithm using ocaml; the ones tagged with "x86" are the execution times of the C++ implementation in the same platform of the Ocaml test; and finally, the ones tagged with "arm" refer to the execution time of the C++ implementation in the PixHawk board.

[1] The instructions to generate the C++ code files that are the output of the use cases experiments are fully detailed in Appendix C.

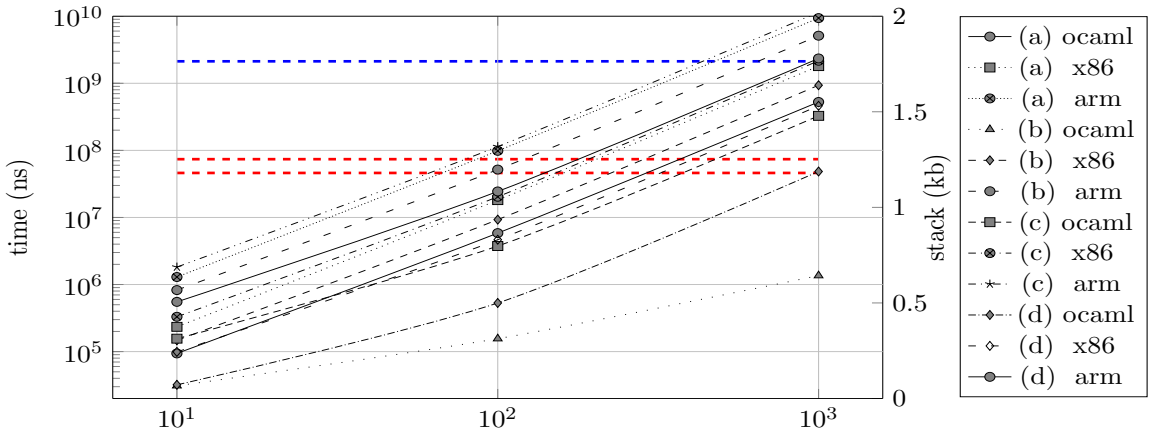


Figure 5.4: Comparison of implementations/architectures

In these experiments, we do not consider more than two nested until operators, which is indeed a common pattern of formulas for the specification of embedded systems. Therefore, we do not have any evidence of how deep nested until operators can be used in a real application scenario.

Experimental results: execution time vs. stack size. Let us first begin with the analysis of the impact in the Ardupilot firmware. The Use Case (1) is composed of several disjunctions, meaning that each branch of the formula can take different execution times. However, the results demonstrate that these formulas are not out of the scope of the previous experiments. The stack usage is 3.4kb for the Use Case (1), and 4.3kb for the formula proposed in the Use Case (2). Based on that, the execution times are on average faster than the worst case considered. Commonly, the monitor increases its execution time as more events are triggered. This means that if the set of events selected for a system is subdivided in different buffers (when possible), then the monitoring will generate lower overheads. However, the impact of the overheads in the Ardupilot is *not* negligible. The overhead generated in the system is 10us/1s for the instrumentation of two sub-tasks, and is 50ms/1s for the monitor (the sub-tasks have periods of 10ms and 5ms respectively). We have also an idle time of about 40% percent. Monitor buffer length is fixed to 100 elements, which is the value obtained according to the pre-calculated time interval required for the formulas under synthesis, and we consider a maximum inter-arrival time of 1ms. The monitors execute with a period of 1s.

Unrecoverable actions. In these use cases, a parachute may be released if a wrong verdict is obtained, or else a safe technique can be deployed, where the multi-copter will spin in order to compensate for a faulty motor. Parachutes are currently used in lightweight aviation to avoid possible unrecoverable mechanical faults, such as motor and propeller failures.

Autopilot Firmware. Ideally, lightweight controller systems should use elastic execution time for tasks, in order to enable the required adaptability for reducing overload situations.

Ardupilot[2] supports several platforms such as AVR, ARM (based on NuttX[3]), and X86 (based on the Linux kernel) [Coombes et al., 2012]. Recently, Ardupilot has adopted non-linear Kalman filters for the *attitude and heading reference system* (AHRS). It is a demanding process that can only be executed in the PixHawk board. For this ARM architecture, two versions are available to perform the same tasks as in imprecise computation definitions. The faster one adopts *direction cosine matrix* (DCM), which is sufficient for the majority of the cases (but is less accurate). The slower version reveals that AHRS can be much better for heavy copters. Ardupilot for the AVR architecture contains several sub-tasks that are scheduled using cyclic scheduling rules. It uses the *Hardware Abstraction Library* (HAL) to communicate with the devices directly, using interrupt-driven routines. However, Ardupilot for PixHawk uses the HAL to communicate with device drivers that are implemented as separate tasks running in NuttX. The RTOS runs a single main task as defined by the AVR architecture, and, instead of using interrupt-driven routines, uses four optional tasks that should be executed at least once each second. These optional tasks have different purposes such as controlling the IO, the UART, and managing timing events and storage (system drivers). The main task contains sub-tasks that execute cyclically in different frequencies ranging from 20hz to 400hz, dictated from the defined cyclic scheduler. The execution rule for sub-tasks is: *based on the predicted WCET, an optional task will execute if there exists available time.*

For construction of a safe autopilot, we are required to ensure time-space isolation. This is crucial for autopilot tasks that have not been formally verified, or are still undergoing testing. To the best of our knowledge none of the currently available autopilot systems for radio control copters have been formally verified. They may well generate absurd values due to *hardware failures*, and are susceptible to *introduced code attacks*, via radio-frequency telemetry links [Moosbrugger et al., 2017].

Summary

Evaluating the proposed theory is of great importance. Formally proving that a real-time scheduler acts as desired, i.e., is correct, is extremely difficulty (it is in many case a combinatorial problem) due to the inherent dependency on time. However, proving

[2] http://copter.ardupilot.org
[3] http://nuttx.org/

it automatically is even more complex and in the majority of the cases it is undecidable (although there are cases where it may be decidable to say if a given settings is schedulable or not according to a given algorithm).

In this chapter, it has been demonstrated that certain classes of real-time scheduling problems can be solved, but not as efficiently as the real-time community could expect. Even though this approach may not scale well, as our results have shown, it points out several issues that would have to be solved in order to increase the applicability of constructing proofs using SMT-based techniques. The positive points are: our results show that it is extremely easy and intuitive to encode scheduling problems in this logic; the approach uses a push button technique to tell us if the scheduling property holds or not, at least in an initial phase (normally saying that a system is unschedulable is close to immediate); and finally the approach mixes offline checking with runtime checks.

In the final part of this chapter, it was shown that monitoring durations even in lightweight platforms such as small embedded systems is feasible and of great importance, in order to avoid possible execution overloads. Overheads are significant depending on the formulas to be monitored. Nevertheless, the push button synthesis allows us to monitor properties in the system for the cases where an event sequence is adopted to log a running application. Acting on the results of monitoring is outside the scope of this work.

Chapter 6

Conclusion and Future Work

RV is a promising technique for making real-time systems (and also other types of systems in general) more reliable and safer. It has been established as a replacement or as complement to static approaches (e.g., model-checking and deductive approaches).

Although RV approaches targeting specifically real-time systems are scarce, they differ from the classic ones. Time bounds and bounded interference are required for explicit time properties. As such, we have developed a new approach for the RV of hard real-time systems, where duration properties play an important role, and incremental evaluation is required. The closest approaches to ours are that of Nickovic and colleagues [Nickovic and Piterman, 2010], who provide synthesis algorithms for MTL specifications, and the work of Pike and colleagues [Pike et al., 2010], who have developed a framework based on a formal stream language, together with a synthesis mechanism that generates monitors. However, none of these previous approaches is sufficiently expressive to allow for reasoning about duration properties, which is the novelty of our work.

The first level of operation of our approach consists of offline analysis for the simplification of formulas by means of quantifier removal techniques; the second is an online evaluation algorithm for RV purposes. We restrict syntactically and semantically the two-valued MTL-$\int$ logic, with a three-valued interpretation. Incremental evaluation allows our technique to handle millions of samples, with formulas containing hundreds of operators.

Another important point is the expressiveness of the logic that has been adopted for this work. Contrary to MTL, which is not sufficiently expressive to deal with explicit durations of propositions/events, our experimental results have revealed that using RMTL-$\int_3$ allows for properties to be specified at the abstraction level of counting time, and to be efficiently synthesized for a platform as small as PixHawk, which is certainly impressive.

Yet, regarding the expressiveness and computing feasibility of timed temporal logics, the unbounded *Since* operator was not considered very relevant in this work, because it requires a full history of a trace. This is not feasible in the context of lightweight real-time embedded systems where resources are scarce. It is known from [Hunter et al., 2013] that for each formula containing the *Since* operator there exists a corresponding formula making use of its dual *Until* operator, which further justifies our exclusive use of the latter operator in this work.

The overall conclusion of our work is that software monitoring techniques, which draw verdicts about timing software faults as well as hardware timing failures, are valid, and may be extremely useful to complement the fault-tolerant mechanisms [Ranjbaran and Khorasani, 2010, Müller and D'Andrea, 2014] that are used for the detection of abnormal mechanical failures.

Additionally, we have described in this thesis an alternative approach to scheduling analysis following a formal based specification of the components of a scheduling hierarchy, and its translation into the SMTLIBv2 language for which we have used the Z3 solver to obtain valid schedules.

6.1 Future work

In terms of future work related to formal languages, it remains to be seen whether extensions of LTL that are strictly more expressive than MTL, such as TPTL [Bouyer et al., 2010] could be used as an alternative for dealing with durations.

Regarding simplification techniques for RV, other efficient mechanisms to reduce the execution time of the monitors as well as the stack usage are required. The shape of the formula impacts severely on its execution time.

Other optimization techniques for synthesis of RMTL-$\int_3$ into SMT problems may be worth exploring. An example is the extension of the synthesis algorithm for interval-based semantics without assuming unit intervals (i.e., intervals of size one), and the consequent repetition of non interleaved symbols. Instead of two intervals $[0, 1[$ and $[1, 2[$ evaluating the symbol a, we have only one interval $[0, 2[$ evaluating a. The theory of strings (word equations) could also be adopted to solve partially the multi-core scheduling problem, instead of the array theory. However, it remains to be seen whether this can be better to explore interleaving of tasks.

Hybrid approaches, in the context of multi-core hard real-time schedulability analysis, can be adopted to treat global scheduling for multi-core systems.

Regarding the synthesis mechanisms, synthesization of RMTL-$\int_3$ into classic *timed automata* (TA) is an option. Although it appears to be unfeasible for RV due to the state explosion problem, encoding time can only be possible if we make use of more expressive classes of automata, such as TA extended with stopwatches [Cassez and Larsen, 2000]. However, the reachability problem for these classes is undecidable, which may imply that no gain should be expected from the point of view of either static analysis or of space complexity for RV purposes.

Regarding the framework, predicting the size of the traces has been considered in this thesis, but more clever solutions should be investigated, for instance along the lines of the idea proposed in [Navabpour et al., 2015]. Instead of estimating the best periods, we could formulate a problem to find the execution pattern that is enough for the application and the monitor. Moreover, we may avoid formulating an optimization problem using linear programming. For that, we might use SMT solvers that we think would be capable to extend the presented schedulability analysis approach to dependent sporadic tasks with monitors.

Regarding the overall thesis, as the rmtld3synth tool is sufficiently mature, other problems could be solved using the proposed techniques. One of them is the monitoring of security threads, throughput, and counting (although not equal, it may be close to MTL with counting [Krishna et al., 2016]). RMTL-$\int_3$ will allow us to deal with a great number of functional properties by adding some syntactic sugar over the duration terms. Even though the word duration refers to time, RMTL-$\int_3$ is able to deal with different units such as space and energy. It is simply a case of meaning.

Appendix A

RV with RMTL-$\int_3$ for C++11

In this section we present a RV framework for embedded RTSs based on the novel RV monitoring model that will be described in Section A.1. The latter contains the constraints/rules from the application side that allow us to synthesize a proper architecture for monitors. These rules are used to configure the target application to be executed in a multi-processor embedded system or over a classic single-processor from the AVR or ARM-M families of embedded processors. The support is given by the RTMLib [De Matos Pedro, 2016] library that allows us to execute monitors in a lock-free and wait-free manner, which is very useful to guarantee deadlock-free RV operation.

Our toolchain is depicted in Figure A.1. As input, we have a set of formulas that will be converted to monitors using a one-to-one correspondence. From these formulas, we generate Ocaml and C++11 source code as well as tests for C++11 implementation that are automatically generated from the Ocaml synthesis, which corresponds to the dependence between both synthesis tools and identified by the dashed arrow. Tests and synthesized monitors are merged and compiled using the gcc toolchain including the support library RMTLib. This binary will run under NuttX OS. Otherwise, the compiled code from the synthesis Ocaml tool is executed in a common x86 operating system.

Operationally, each monitor can share resources (*e.g.*, memory and processors) with other monitors or may execute in isolation (using its own processor and memory partition), which is part of the specification of the RV monitoring model. The monitors have different execution rules that may change at execution time, and rules for their operation.

- Execution rules are step-based (for iterative/tail recursive monitors; for an arbitrary number $n \in \mathbb{N}$ of execution steps), symbol-based (for explicit symbol consumption in automata formalisms), time-based (a timed bound in discrete execution time for execution of general purpose monitors). Based on this we can change the execution

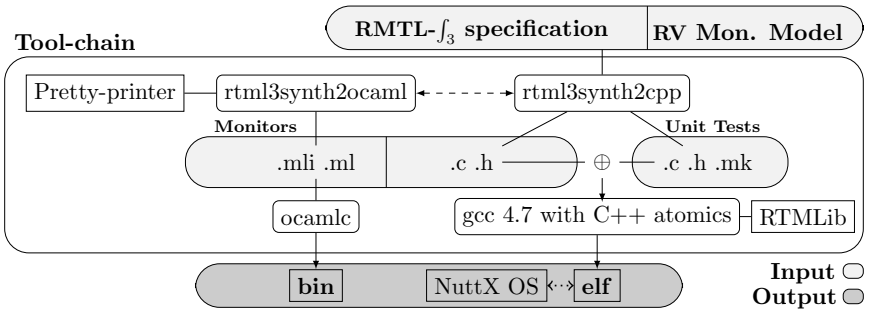


Figure A.1: Tool-chain overview

of the monitor at runtime in a dynamic way (a feature provided by RTMLib).

- Operation rules are time-triggered or event-triggered; the idea is to generate runtime verifiers depending of the target RTS. The modes of operation/execution are assigned according to the RV model.

For hard RTS, we use the step-based rule combined with a time-triggered rule. Note that there is no explicit architecture for monitoring, and different RV rules produce different monitor architectures, depending on the target systems and the provided RV monitoring model.

Synthesis Algorithm Refinement. The evaluation algorithm proposed for RMTL-$\int_3$ in the Chapter 3 uses functional programming language features such as *pattern matching* and *higher-order functions*, in particular *fold* operations.

Let $\mathbf{K}$ be a set of sequences κ, Υ a set of logic environments υ, and $\mathbb{R}_{\geq 0}$ the domain of a time instant t (analogous to the model (κ, υ, t)). Let us first consider the lambda functions, as already defined in the Chapter 3, such as $\mathsf{Compute}_{(\vee)} :: (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \to \Phi^3 \to \Phi^3 \to \mathbb{B}_3$, $\mathsf{Compute}_{(\neg)} :: (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \to \Phi^3 \to \mathbb{B}_3$, $\mathrm{Compute}_{(U_<)} :: (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \to \mathbb{R}_{\geq 0} \to \Phi^3 \to \Phi^3 \to \mathbb{B}_3$, and $\mathsf{Compute}_{(\int)} :: (\mathbf{K} \times \Upsilon) \to \mathbb{R}_{\geq 0} \to \mathbb{R} \to \Phi^3 \to \mathbf{D}$, that evaluate formula schemes of the form $\psi_1 \vee \psi_2$, $\neg\psi$, $\psi_1 \,\mathrm{U}_{<\gamma}\, \psi_2$, and $\int^{\eta} \psi$, respectively. Note that $(\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0})$ is a model (consisting of a sequence in $\mathbf{K}$, a logic environment in Υ, and a time instant in $\mathbb{R}_{\geq 0}$), $\mathbf{D}$ the set $\mathbb{R}_{\geq 0} \cup \{\bot_{\mathbb{R}}\}$, Φ^3 is a set of three-valued formulas, $\mathbb{B}_3$ is the set of three-values $\{\mathsf{tt}, \mathsf{ff}, \bot\}$, and $\mathbf{B}_4$ is a four-valued set defined by $\mathbb{B}_3 \cup \{\mathfrak{r}\}$, where $\mathfrak{r}$ is the fourth symbol of the four-valued set. Pattern matching features are currently not included in imperative programming languages such as C++11. Henceforth, and for the sake of compatibility with C++11, we adapt that algorithm as follows:

– the pattern matching constructions are statically erased and fully encoded into the generated monitors;

– the fold functions are encoded as *iterators* over the structure of traces;

– the remaining functions are encoded as `C++11` *lambda functions*.

Pattern matching is simplified over the inductive structure of the formulas. For instance, the formula $a \rightarrow \int^{10} b$ is implemented without pattern matching by composition over the structure of the formula. For that, we need to define some new `C++11` lambda functions such as $compute_p :: \mathbf{P} \rightarrow (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \rightarrow \mathbb{B}_3$, $compute_\neg :: ((\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \rightarrow \mathbb{B}_3) \rightarrow (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \rightarrow \mathbb{B}_3$, $compute_\int :: \mathbb{R} \rightarrow ((\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \rightarrow \mathbb{B}_3) \rightarrow (\mathbf{K} \times \Upsilon) \rightarrow \mathbb{R}_{\geq 0} \rightarrow \mathbf{D}$, and

$$\begin{aligned} compute_\vee :: \quad & ((\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \rightarrow ((\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \rightarrow \mathbb{B}_3) \rightarrow \mathbb{B}_3) \rightarrow \\ & (((\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \rightarrow \mathbb{B}_3) \rightarrow (\mathbf{K} \times \Upsilon) \rightarrow \mathbb{R}_{\geq 0} \rightarrow \mathbf{D}) \rightarrow \\ & (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \rightarrow \mathbb{B}_3. \end{aligned}$$

Note that they encode the pattern matching (all required combinations for a given formula) instead of accepting RMTL-$\int_3$ formulas as input arguments. The generated function that corresponds to $a \rightarrow \int^{10} b$ is then the lambda function

$$\lambda m.\ compute_\vee\ (compute_\neg\ (compute_p\ a))\ \Big(compute_\int\ 10\ (compute_p\ a)\Big)\ m$$

where m is the model defined in `C++11` as `TraceIterator<int> iter`, `struct Environment env`, and `timespan t`. Note that $\lambda x.\ fun$ is defined in `C++11` as the expression `[](x){fun}`.

Let us now focus on the U operator. Porting to `C++11` the function $\text{Compute}_{(U_<)}$, responsible for the synthesis of the until operator, requires defining a number of auxiliary `C++11` functions. As an example, the function $\text{ev}_{\text{al}}^{\text{fold}} :: (\mathbf{K} \times \Upsilon \times \mathbb{R}_{\geq 0}) \rightarrow \Phi^3 \rightarrow \Phi^3 \rightarrow \mathbf{K} \rightarrow \mathbf{B}_4$, as provided in the original RMTL-$\int_3$ evaluation algorithm, is defined in `C++11` as shown in Listing 1. We remark that the synthesized function $(\text{ev}_{\text{al}}^{\text{fold}}\ (\kappa, \upsilon, t)\ \phi_1\ \phi_2\ \varkappa)$ is originally defined by

$$fold\ \Big(\lambda v\ (p, (i, t')) \rightarrow \text{ev}_{\text{al}}^{\text{b}}\ (\kappa, \upsilon, t' - \epsilon)\ \phi_1\ \phi_2\ v\Big)\ \mathfrak{r}\ \varkappa,$$

where ϕ_1 and ϕ_2 are formulas that were statically coded as the `C++11` lambda functions $\text{ev}_{\text{al}}^{\text{b}}$ (of which there exist as many as there are occurrences of until operators, since each one contains different formulas), $\varkappa$ is the original trace sequence that is mapped into the iterator *iter* of Listing 1, and i is the lower bound of the interval (i, t'), ϵ is the minimum precision of a float, and $\mathfrak{r}$ is a proper mark for release if the until evaluation gives us an unknown value, identified in `C++11` by `FV_SYMBOL`, respectively. The operators $U_<$, $<$, and duration terms $\int^\eta \varphi$ may now be fully implemented using the `C++11` lambda functions.

```
auto eval_fold = []( struct Environment env, timespan t, TraceIterator<int> iter) > four_valued_type
{
  return std::accumulate
 (
   iter.begin(), iter.end(), pair<four_valued_type, timespan>( FV_SYMBOL, t ),
   [&env, eval_b]( const pair<four_valued_type, timespan> a, Event<int> e )
   {
     return make_pair( eval_b( env, a.second, a.first ), a.second + e.getTime() );
   }
  ).first;
};
```

Listing 1: $\text{ev}_{\text{al}}^{\text{fold}}$ synthesis in C++11

The existential operator does not need to be treated since we assume the existence of a simplification algorithm that decomposes a quantified formula into a non quantified formula. The output of this tool is a monitor written in the C++11 programming language and composed by several source files, and the input is a configuration file containing an RMTL-$\int_3$ formula to be synthesized. The `rmtld3synth` synthesis tool for these operators, written in the Ocaml programming language [The OCaml Development Team, 2013] is fully described in [De Matos Pedro, 2018]. The reader is referred to the example in Appendix B for further details and a worked out example.

A.1 RV Monitoring Model

In this section we describe how monitors are linked to buffers and tasks via the specialized *RunTime Embedded Monitoring Library* (RTMLib), and then discuss how timing guarantees are enforced in practice by the adopted hierarchy of monitors.

Linking monitors with RTMLib

Monitors are executed in a simple embedded monitoring framework which we named the RTMLib [De Matos Pedro, 2016]. These monitors use circular buffers as the data structure to hold a trace, and they have a certain periodicity. The framework ensures that monitors retrieve events from circular buffers respecting their partial order, in a lock- and wait-free manner. Note that several buffers are used in a composition as described in [Nelissen et al., 2015] for the reference architecture; more details on the implementation of RTMLib can be found in the documentation in [De Matos Pedro, 2016]. Monitors execute as higher-priority tasks and are constantly interfering with the application. However, such interference is predictable and constant, since each monitor can execute in constant time that depends on the structure of the formula.[1]

[1] By constant time we mean that a monitor executes the same number of CPU cycles at each invocation.

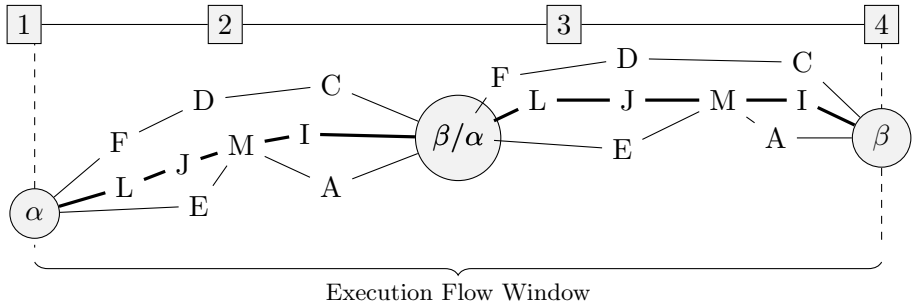


Figure A.2: Flow graph of the system enabled events defined in a time window.

Knowledge of the length of the circular buffers is required at compile time, and for that we define a *bound* over temporal formulas, allowing us to determine a map from time to event size. The calculation of temporal bounds for formulas of RMTL-$\int_3$ is then achieved by a recursive algorithm that traverses the inductive structure of the formulas. We now give two examples of the calculation of an upper bound for a given formula, and the construction of a flow graph for a given time window.

Example 20. *Let us consider a trace and the formula* $a\ \mathrm{U}_{<10}\ (b\ \mathrm{U}_{<10}\ c)$*, containing propositions* a, b, c *evaluated at time* $t = 0$*. Based on the semantics of temporal operators we achieve the timing bounds* $t \in]0, 10[$ *and* $t \in]0, 20[$*, respectively. These time bounds are intervals where the truth values resulting from the evaluation of formulas may change. By the semantic nature of temporal operators, we know that for any* $t \notin]0, 10[\cup]0, 20[$ *the truth value is maintained constant, which gives us the desired bound for changes of the evaluation value.*

Example 21. *In order to estimate the amount of time required from the system under observation to couple monitors in a safe manner, we can use a pessimistic approach based on the assumption of a maximum inter-arrival time of events in the system, or we can pre-compute the flow graph of the application. Based on these, we are able to infer how many events will be triggered in a certain time interval. To exemplify the specific case of the latter, we define a time window given by a certain formula using the previous approach. Then, we create a flow graph of the entire system and fix the starting point of the system as depicted in the partial flow pattern of the events (ranging from symbol A to M) under monitoring in the Figure A.2. From label* α *to* β*, where* α *corresponds to the beginning of the execution and* β *corresponds to the end of the execution, we have the flow of the main task composed by three paths (the task that manages the autopilot controller), and from label 1 to 4, we have the optional task (a time-triggered task for device drivers execution that need to execute at least 1 time in a second). The optional task has two times the period of the main task. In summary, we have at most four events between* α *and* β *and the optional task executes twice between them. The figure also depicts the dependencies of*

events, and allows us to estimate the required relative time for some events.

Altogether, these examples combine temporal settings of the monitors and the system itself: the first one give us the amount of time that we need to wait for a verdict (minimum time granularity); the second one helps us to find the period for a monitor based on the time behavior of the system under monitoring as well as to estimate the WCET of the monitor (i.e, the time complexity times a constant).

Appendix B

`rmtld3synth` tool User's Guide

The `rmtld3synth` synthesis tool is able to automatically generate monitors based on the formal specifications written in RMTL-$\int_3$. Polynomial inequalities are supported by this formalism as well as the most common operators of temporal logics. Furthermore, quantification is also considered in the language of RMTL-$\int_3$ as a means to facilitate the decomposition of the quantified formulas into several monitoring conditions.

We will now present an overview of the typical process for generating monitors for Ocaml and `C++11` languages using this tool, together with a running example of a simple monitoring case generation. We begin by the running example, present the generated monitors, and show how to configure the RV monitoring model to couple with the system.

Consider the formula

$$(a \rightarrow ((a \vee b)\, \mathrm{U}_{<10}\, c)) \wedge \int^{10} c < 4 \tag{B.1}$$

that intuitively describes that given an event a, b occurs until c and, at the same time, the duration of b shall be less than four time units over the next 10 time units. For instance, a trace that satisfies this formula is

$$(a, 2), (b, 2), (a, 1), (c, 3), (a, 3), (c, 10).$$

From `rmtld3synth2ocaml` tool, we have synthesized the formula's example into the code of the Listing 3. For that, we have used the command in the Listing 1.

```
./rmtld3synth --synth-ocaml --input-latexeq "(a \rightarrow ((a \
   lor b) \until_{<10} c)) \land \int^{10} c < 4"
```

Listing 1: Utilized shell command for the Equation B.1

Next, we can also generate `C++11` monitors by replacing `--synth-ocaml` with `--synth-cpp11`.

The outcome is the monitor illustrated in the Listings 4 and 5. To use those monitors, we need to define a trace for Ocaml reference as in the Listing 2.

```
module OneTrace : Trace = struct let trc = [("a",(0.,2.));("b
    ",(2.,4.));("a",(4.,5.));("c",(5.,8.));("a",(8.,11.));("c
    ",(11.,21.))] end;;

module MonA = Mon0(OneTrace);;
```

Listing 2: Ocaml's reference code for monitor instantiation

For the Ocaml language, experimental integration with RTMLib is available. However, we do not describe it here, but refer the reader for the examples in `rmtld3synth`'s repository [1]. For `C++11` we will now briefly describe how it is performed. Given the verbosity of the generated code, we have removed the conjunction including the duration inequality, and used instead the simple formula

$$\int^{10} c < 4.$$

Now, we describe the settings for constructing the RV monitoring model.

Overview of the configuration settings. The settings for `rmtld3synth` tool are defined using the syntax

```
(<setting_id> <bool_type | integer_type | string_type>)
```

where `|` distinguishes between the supported types of arguments such as Boolean, integer or string, and `setting_id` is a string containing the name of the setting to which values are assigned. An example of a set of possible settings for the tool is given in the first five lines of Listing 6. We now briefly describe the purpose of each of the setting entries present in Listing 6:

- `gen_tests` sets the automatic generations of test cases (to be used as a demo in the described illustration below).

- `gen_concurrency_tests` constructs tests for testing lock- and wait-free monitors executing concurrently.

- `gen_unit_tests` constructs tests for `C++11` synthesis using the Ocaml source code as an oracle.

[1] Available at https://github.com/anmaped/rmtld3synth/tree/v0.3-alpha, version 0.3-alpha.

```
open List
open Rmtld3

module type Trace = sig val trc : trace end
module Mon0  ( T : Trace  ) = struct
let compute_uless gamma f1 f2 k u t =
  let m = (k,u,t) in
  let eval_i b1 b2 =
    if b2 <> False then b3_to_b4 b2 else if b1 <> True && b2 = False then b3_to_b4 b1 else
        Symbol
  in

  let eval_b (k,u,t) f1 f2 v =
    if v <> Symbol then v else eval_i (f1 k u t) (f2 k u t)
  in

  let eval_fold (k,u,t) f1 f2 x =
    fst (fold_left (fun (v,t') (prop,(ii1,ii2)) > (eval_b (k, u, t') f1 f2 v, ii2)) (Symbol,t)
        x)
  in

  if not (gamma >= 0.) then
    raise  (Failure "Gamma_of_U_operator_is_a_non negative_value")
  else
  begin
    let k,_,t = m in
    let subk = sub_k m gamma in
    let eval_c = eval_fold m f1 f2 subk in
    if eval_c = Symbol then
      if k.duration_of_trace <= (t +. gamma) then Unknown else ( False ) else b4_to_b3 eval_c
  end


let compute_tm_duration tm fm k u t =
  let dt = (t,tm k u t) in

  let indicator_function (k,u) t phi = if fm k u t = True then 1. else 0. in
  let riemann_sum m dt (i,i') phi =
    (* dt=(t,t') and t in ]i,i'] or t' in ]i,i'] *)
    count_duration := !count_duration + 1 ;
    let t,t' = dt in
    if i <= t && t < i' then
      (* lower bound *)
      (i' .t) *. (indicator_function m t phi)
    else (
      if i <= t' && t' < i' then
        (* upper bound *)
        (t' .i) *. (indicator_function m t' phi)
      else
        (i' .i) *. (indicator_function m i phi)
    ) in
  let eval_eta m dt phi x = fold_left (fun s (prop,(i,t')) > (riemann_sum
  m dt (i,t') phi) +. s) 0. x in
  let t,t' = dt in
  eval_eta (k,u) dt fm (sub_k (k,u,t) t')

  let env = environment T.trc
  let lg_env = logical_environment
  let t = 0.
  let mon = (fun k s t > b3_not ((fun k s t > b3_or ((fun k s t > b3_not ((fun k s t > b3_or
       ((fun k s t > b3_not ((fun k s t > k.evaluate k.trace "a" t) k s t)) k s t) ((
      compute_uless 10. (fun k s t > b3_or ((fun k s t > k.evaluate k.trace "a" t) k s t) ((
      fun k s t > k.evaluate k.trace "b" t) k s t)) (fun k s t > k.evaluate k.trace "c" t)) k
      s t)) k s t)) k s t) ((fun k s t > b3_not ((fun k s t > b3_lessthan ((
      compute_tm_duration (fun k s t > 10.) (fun k s t > b3_or ((fun k s t > k.evaluate k.
      trace "c" t) k s t) ((fun k s t > k.evaluate k.trace "d" t) k s t))) k s t) ((fun k s t
       > 4.) k s t)) k s t)) k s t)) k s t)) env lg_env t
end
```

Listing 3: Generated Ocaml monitor

```
#ifndef _MON0_COMPUTE_H_
#define _MON0_COMPUTE_H_
#include "rmtld3.h"

auto _mon0_compute = [](struct Environment &env, timespan t) mutable -> three_valued_type {
     return [](struct Environment env, timespan t) -> three_valued_type { auto tr1 = [](struct
     Environment env, timespan t) -> duration {

  auto eval_eta =  [](struct Environment env, timespan t, timespan t_upper, TraceIterator< int >
       iter) -> duration
  {
    auto indicator_function = [](struct Environment env, timespan t) -> duration {
      auto formula = [](struct Environment &env, timespan t) mutable -> three_valued_type { auto
           sf1 = [](struct Environment &env, timespan t) mutable -> three_valued_type { return
          env.evaluate(env, 2, t); }(env,t); auto sf2 = [](struct Environment &env, timespan t)
          mutable -> three_valued_type { return env.evaluate(env, 1, t); }(env,t); return b3_or
          (sf1, sf2); }(env, t);

      return (formula == T_TRUE)? std::make_pair (1,false) : ( (formula == T_FALSE)? std::
          make_pair (0,false) : std::make_pair (0,true)) ;

    };

    auto lower = iter.getLowerAbsoluteTime();
    auto upper = iter.getUpperAbsoluteTime();
    timespan val1 = ( t == lower )? 0 : t   lower;
    timespan val2 = ( t_upper == upper )? 0 : t_upper   upper;
    auto cum = lower;

    return std::accumulate(
      iter.begin(),
      iter.end(),
      std::make_pair (make_duration (0, false), (timespan)lower), (duration starts at 0)
      [&env, val1, val2, &cum, t, t_upper, indicator_function]( const std::pair<duration,
          timespan> p, Event< int > e )
      {
        auto d = p.first;
        auto t_begin = cum;
        auto t_end = t_begin + e.getTime();
        cum = t_end;
        auto cond1 = t_begin <= t && t < t_end;
        auto cond2 = t_begin <= t_upper && t_upper < t_end;
        auto valx = ((cond1)? val1 : 0 ) + ((cond2)? val2 : 0);
        auto x = indicator_function(env, p.second);

        return std::make_pair (make_duration (d.first + (x.first * ( e.getTime()   valx )), d.
            second || x.second), p.second + e.getTime());
      }
    ).first;
  };

  auto sub_k = []( struct Environment env, timespan t, timespan t_upper) -> TraceIterator< int >
  {
    TraceIterator< int > iter = TraceIterator< int > ( env.trace, env.state.first, 0, env.state.
        first, env.state.second, 0, env.state.second );
    // to use the iterator for both searches we use one reference
    TraceIterator< int > &it = iter;

    ASSERT_RMTLD3( t == iter.getLowerAbsoluteTime() );
    auto lower = env.trace->searchIndexForwardUntil( it, t);
    auto upper = env.trace->searchIndexForwardUntil( it, t_upper   1 );
    it.setBound(lower, upper);

    return it;
  };

  auto t_upper = t + make_duration(10.,false).first;

  return eval_eta(env, t, t_upper, sub_k(env, t, t_upper));

  }(env,t);

  auto tr2 = make_duration(4.,false);
  return b3_lessthan (tr1, tr2);
}(env,t);};
#endif //_MON0_COMPUTE_H_
```

Listing 4: Generated C++11 monitor

```
#ifndef MONITOR_MON0_H
#define MONITOR_MON0_H
#include "Rmtld3_reader.h"
#include "RTML_monitor.h"
#include "mon0_compute.h"
#include "mon1.h"

class Mon0 : public RTML_monitor {
private:
  RMTLD3_reader< int > trace = RMTLD3_reader< int >( __buffer_mon1.getBuffer(), 0. );
  struct Environment env;

protected:
  void run(){
    three_valued_type _out = _mon0_compute(env,0);
    DEBUG_RTEMLD3("Veredict:%d\n", _out);
  }

public:
  Mon0(useconds_t p): RTML_monitor(p,SCHED_FIFO,50), env(std::make_pair (0, 0), &trace,
       __observation) {}
};
#endif //MONITOR_MON0_H
```

Listing 5: Generated C++11 monitor header

- `buffer_size` sets the static size of the buffer to be used (`rmtld3synth` tool can change it if required by some constraints).

- `minimum_inter_arrival_time` establishes the minimum inter-arrival time that the events can have. It is a very pessimistic setting but provides some information for static checking.

- `maximum_period` sets the maximum interval between two consecutive releases of a task's job. It has a correlation between the periodic monitor and the minimum inter-arrival time. It provides static checks according to the size of time-stamps of events.

- `event_type` provides the type for dealing with events (commonly is a class parameter).

- `event_subtype` provides the type for the event data. In that case, it is an identifier that can distinct 255 events. However, if more events are required, the type should be modified to *uint32_t* or greater. The number of different events versus the available size for the identifier is also statically checked.

- `cluster_name` identifies the set of monitors. It acts as a label for grouping monitor specifications.

Writing formulas in RMTLD3 The formulas 'm_simple' and 'm_morecomplex' follow the same syntax defined in this section. For setting a periodic monitor, we use

```
(gen_tests true)
(minimum_inter_arrival_time 102)
(maximum_period 2000000)
(event_subtype uint_8)
(cluster_name monitor_set1)

(m_simple 1000000 (Or (Until 200000 (Prop A) (Prop C)) (Prop B)))
(m_morecomplex 500000 (Or (Until 200000 (Prop set_off) (Or (Until 200 (Prop A) (Prop C)) (Prop B))) (
    Prop B)))
```

Listing 6: The default configuration file.

```
type var_id = string with sexp
type prop = string with sexp
type time = float with sexp
type value = float with sexp

type formula =
          True of unit
        | Prop of prop
        | Not of formula
        | Or of formula * formula
        | Until of time * formula * formula
        | Exists of var_id * formula
        | LessThan of term * term
and term =
          Constant of value
        | Variable of var_id
        | FPlus of term * term
        | FTimes of term * term
        | Duration of term * formula
with sexp

type rmtld3_fm = formula with sexp
type rmtld3_tm = term with sexp
type tm = rmtld3_tm with sexp
type fm = rmtld3_fm with sexp
```

Listing 7: The inductive type.

(m_usecase1 <period> (<monitor sexpr>)). They are formatted as a symbolic expression. The type in Ocaml is according to the Listing 7.

Appendix C

RTMLib

The *RunTime Embedded Monitoring Library* (RTMLib) is a library that has been developed with the purpose of runtime monitoring of real-time embedded systems. RTMLib is based on lock-free ring buffer FIFO queues for managing the information from events that are registered in buffers. The library is supported in both ARM and x86 platforms. Efficient architectures can be developed based on lock-free enqueue and dequeue primitives over trace sequences containing time stamped events. Synchronization primitives for dequeueing operations allow different readers to progress synchronously over the target instantiated buffers. Buffers are implemented with different timestamps, depending of the architecture. For ARM it uses 32bit values to save memory, and for x86 it uses 64bit timestamps.

C.1 Usage of RTMLib

C.1.1 Instantiating buffers

Buffers are resources shared between the SUO and the monitors. Buffers contain time-stamped event sequences that inform monitors of the changes in the state of the SUO. RTMLib requires at least one global buffer available for the instrumentation of the SUO, and that at the linking phase of the compilation shall provide the address of the buffer for external monitors to make use of it. We define a "interface.h" header file that serves as the interface header to be used by both the SUO and the monitors. The code of the Listing 1 exemplifies this requirement.

Note that this code, `uint8_t` could be used to represent events identified as integers ranging from 0 to 255 only. Other types such as `uint16_t` and `uint32_t` could also be

```
#include "RTEML_buffer.h"

extern void __start_periodic_monitors();

// defining one buffer with size 100 of type uint8_t
extern RTEML_buffer<uint8_t, 100> __buffer_monitor_set1;

#define EV_C 3
#define EV_A 4
#define EV_set_off 5
#define EV_B 1
```

Listing 1: interface.h sample file.

```
#include "M_morecomplex.h"
#include "M_simple.h"
#include "RTEML_buffer.h"

RTEML_buffer<uint8_t, 100> __buffer_monitor_set1;

M_morecomplex mon_m_morecomplex(__buffer_monitor_set1, 500000);
M_simple mon_m_simple(__buffer_monitor_set1, 1000000);

void __start_periodic_monitors()
{
  if (mon_m_morecomplex.enable()) {::printf("ERROR\n");}
  if (mon_m_simple.enable()) {::printf("ERROR\n");}
}
```

Listing 2: interface.cpp sample file.

used to increase the number of different kinds of events that can be considered. However, strings and classes are discouraged as they bring extra memory space overhead that, in the extreme, can compromise the whole implementation of adding monitors into the target SUO[1].

The instantiation of buffers and monitors together shall follow along the lines of the programming structure used in the code listed below. Note, however, that is not mandatory to instantiate the buffer with the monitors as the Listing 2 describes. The `M_simple.h` header defines a monitor according to what is described in the next paragraph. The `M_morecomplex.h` header defines another monitor that shares the buffer `__buffer_monitor_set1`, and `__start_periodic_monitors` is the procedure used to initialize both monitors.

C.1.2 Developing a simple Monitor

We now show how to construct a simple monitor based on `RTEML_monitor` class. First, the `RTEML_monitor` class enables monitors to execute at a certain periodicity. The class is initialized using some arguments such as the period, the scheduler policy, and the priority. The scheduler policies and priorities are commonly OS dependent. For instance, in Windows Embedded 10 x86, we only have available the `SCHED_FIFO` policy in pthreads-

[1]The natural alternative is to map these events in a hash table to save memory space.

```
#include "interface.h"

class M_simple : public RTEML_monitor {

  private:
    RTEML_reader<int> __reader = RTEML_reader<int>(__buffer_monitor_set1.getBuffer());

  protected:
    void run(){
      ::printf("Body of the monitor.");
    }

  public:
    M_simple(useconds_t p): RTEML_monitor(p,SCHED_FIFO,5) {}

};
```

Listing 3: monitor.h sample file.

win32, and priorities can be negative and range from -15 (lowest) to 15 (highest). Zero is the normal priority.

For fully Posix compliant OS, the priorities are non negative and several policies such as `SCHED_RR` (round robin) and `SCHED_OTHER` exist. In case of NuttX OS, we have the same policies. The class `M_simple` is defined in the Listing 3. This monitor will display the string "Body of the monitor." several times with a period of `p` useconds. Lets replace the 'run' procedure with a consumer procedure as exemplified in the paragraph below.

Consumer procedure. The consumer process is exemplified using one lambda function. It fits the required interface defined in `RTEML_monitor` for the procedure `run`. The body of the function initializes an object of type `RTEML_reader<int>` that will be used as the consumer for the lock-free buffer. The procedure `dequeue()` peek a tuple containing an event of type `Event<int>`, where the template typename is the type of the expected identifier of the event, and a time-stamp. Note that the dequeue is local to the reader, does not affect the global buffer, and can be synchronized using a certain time-stamp. However, to get a global dequeue of a certain event, we shall share the same reader among the tasks. The consumer is defined in the Listing 4, where the variable `tmpEvent` stores the dequeued event, where the methods `getTime()` and `getData()` return the time-stamp and the event identifier, respectively.

Producer procedure for Monitors Lets construct a producer for the lock-free ring buffers. First, we initialize the object `__writer` of the type `RTEML_writer<int>`. Then, we enqueue a value of type `int` to the buffer that accepts events of the type `Event<int>`, and finally print the buffer to the *stdout* for debugging purposes. The code is described in the Listing 5.

```
auto consumer = [](void *) > void*
{
  static RTEML_reader<int> __reader = RTEML_reader<int>(__buffer_monitor_set1.getBuffer());
  Event<int> tmpEvent;

  std::pair<state_rd_t,Event<int> &> rd_tuple = __reader.dequeue();
  tmpEvent = rd_tuple.second;
  ::printf("event_out: %lu, %d code: %d\n", tmpEvent.getTime(), tmpEvent.getData(), rd_tuple.first);

  return NULL;
};
```

Listing 4: Example of a consumer using lambda functions.

```
auto producer = [](void *) > void*
{
  static RTEML_writer<int> __writer = RTEML_writer<int>(__buffer_monitor_set1.getBuffer());

  __writer.enqueue(1);

  __buffer_monitor_set1.debug();
  return NULL;
};

__task producer_A = __task(producer, 0, SCHED_FIFO, 100000);
```

Listing 5: Example of a producer using lambda functions.

Note that `__task` is an helper used to construct the data descriptor of one task. It inputs the function pointer, the priority, the scheduler policy, and the period. `100000` means $\frac{1}{10}$ seconds.

Appendix D

Inequality Translation Correctness Proofs

The following proofs are related with the Lemmas 5 and 6 that were enunciated in the Chapter 3. Let us now introduce some required definitions and one auxiliar Lemma 11 before introducing the main proofs. Let us assume in this appendix that every formula ϕ_i is in DNF_3.

Definition 15. Let $f_\phi(X, Y, Z)$ be a shorthand for $(X \rightarrow Y) \wedge (\neg X \rightarrow Z)$, where X, Y and Z are formulas in RMTL-$\int_3$.

Lemma 11. *Let ϕ be a finite formula in RMTL-$\int_3$ containing propositions and inequalities composed by rigid terms, and $n > 0$ the number of inequalities of ϕ with $n \in \mathbb{N}$. Then, there is an equivalent formula resulting from the application of both A4 and A5 at most $2^n - 1$ times, and containing 2^n disjunctions.*

Proof of Lemma 11. Straightforward induction over n. Let b be the function recursively defined by $f(m) = 1 + f(m-1) + f(m-1)$ with $f(0) = 0$, where $f(m)$ denotes the number of resulting disjunctions, and $m = \lceil \log_2 x \rceil$, where x is the number of applied axioms. Note that this function is structurally similar to the shape of A4 and A5 after applying the simplification of implications to DNF_3 of the form $(X \wedge \phi_1) \vee (Y \wedge \phi_2)$, where ϕ_1 and ϕ_2 are arbitrary sub-formulas that can be finitely expanded. We want to show that $f(n) + 1 = 2^n$.

Base case: $f(1) + 1 = 2^1$.

Inductive case: $f(n) + 1 = 2^n$

$$
\begin{aligned}
f(n) &= 1 + f(n-1) + f(n-1) \\
f(n) &= 2^{n-1} + 2^{n-1} - 1 \\
f(n) &= 2^1 \cdot 2^{n-1} - 1 \\
f(n) &= 2^n - 1 \\
f(n) + 1 &= 2^n
\end{aligned}
$$

□

Lemma 12. *Let ϕ be a finite formula in RMTL-$\int_3$ containing inequalities, and $n > 0$ the number of inequalities of ϕ with $n \in \mathbb{N}$. There is an equivalent formula resulting from the application of both A6 and A7 at most n times and contain m disjunctions.*

Let us now recall the Lemma 5.

Lemma 5. *Let ϕ^1, ϕ^2 be two formulas in RMTL-$\int_3$ and consider the formula $\phi^1 \, U \, \phi^2$. Then, there exists an equivalent formula where every until operator is free of inequalities or only contains equalities of the form $x = \int^\eta \varphi$.*

Proof of Lemma 5. By induction along the structure of the formulas ϕ^1 and ϕ^2.

- **Base cases**:

  1. ϕ^1, ϕ^2 do not contain inequalities:

     The proof is straightforward. First, we apply A4 and we get $\phi_r \rightarrow \phi_1 \vee \phi_3 \; U \; \phi_2$ or $\neg\phi_r \rightarrow \phi_1 \, U \, \phi_2$ and ϕ_3 equals to false. Since both disjunctions are equal, we get $\phi_1 \, U \, \phi_2$, and by definition that $\mathbf{f}_1^{<}(\phi_2) := true$, $\mathbf{f}_1^{<}(\phi_1) := true$, $\mathbf{f}_1^{\not<}(\phi_1) := \phi_1$, and $\mathbf{f}_1^{\not<}(\phi_2) := \phi_2$. Therefore, Property 1 holds with $true \wedge true \wedge \phi_1 \; U \; \phi_2$ equal to X_1.

  2. ϕ^1, ϕ^2 contain inequalities involving propositions:

     Let ϕ_1 be equal to $(a_1 \wedge \cdots \wedge p_{a,1}) \vee \cdots \vee (a_l \wedge \cdots \wedge p_{a,l})$, and ϕ_2 equal to $(b_1 \wedge \cdots \wedge p_{b,1}) \vee \cdots \vee (b_l \wedge \cdots \wedge p_{b,l})$, and a_i,b_i be inequalities composed by rigid terms with $i, j \in \mathbb{N}$.

     For the sake of simplicity, we denote $\phi_{a1} := (\cdots \wedge p_{a,1}) \vee \cdots \vee (a_l \wedge \cdots \wedge p_{a,l})$, $\phi_{a2} := \cdots \vee (a_l \wedge \cdots \wedge p_{a,l})$, $\phi_{b1} := (\cdots \wedge p_{b,1}) \vee \cdots \vee (b_l \wedge \cdots \wedge p_{b,l})$, and $\phi_{b2} := \cdots \vee (b_l \wedge \cdots \wedge p_{b,l})$.

Applying A4 and A5 for the formulas ϕ_1 and ϕ_2, we have the formula

$$(a_1 \wedge b_1 \wedge \phi_{a1}\ U\ \phi_{b1}) \vee$$
$$(a_1 \wedge \neg b_1 \wedge \phi_{a1}\ U\ \phi_{b2}) \vee$$
$$(\neg a_1 \wedge b_1 \wedge \phi_{a2}\ U\ \phi_{b1}) \vee$$
$$(\neg a_1 \wedge \neg b_1 \wedge \phi_{a2}\ U\ \phi_{b2}).$$

From Lemma 11 we know that there are so many disjunctions as the 2^n, where n is the number of inequalities contained jointly in ϕ_1 and ϕ_2.

From the shape of A4 and A5, we see that at most four formulas ϕ_1, ϕ_2, ϕ_3, and ϕ_r are involved. For A4, we get by definition $(\phi_{m1} \wedge \phi_{m2}\ U\ \phi_{m3})$, where $\phi_{m1} := \mathbf{f}_i^{<}(\phi_3) \wedge \mathbf{f}_i^{<}(\phi_1 \vee (\phi_r \wedge \phi_2))$, $\phi_{m2} := \mathbf{f}_i^{\not<}(\phi_1 \vee (\phi_r \wedge \phi_2))$, and $\phi_{m3} := \mathbf{f}_i^{\not<}(\phi_3)$. For A5 the same scheme is followed. Both resulting formulas $(\phi_{m1} \wedge \phi_{m2}\ U\ \phi_{m3})$ and $\phi_{m1}\ U\ (\phi_{m2} \wedge \phi_{m3})$ indicate that three formulas $\phi_{m1}, \phi_{m2}, \phi_{m3}$ are required. Since for all i such that $0 < i \leq n$ there exist functions $\mathbf{f}_i^{<}(\phi_1)$, $\mathbf{f}_i^{\not<}(\phi_2)$, $\mathbf{f}_i^{\not<}(\phi_1)$, and $\mathbf{f}_i^{\not<}(\phi_2)$ that map inequalities for each disjunction, then Property 1 holds for $n = m$.

For the cases of ϕ^1 or ϕ^2 containing exclusively inequalities with propositions the proof is similar.

3. ϕ^1, ϕ^2 contain inequalities with duration terms:

   The proof begins as similar as the proof above and then proceeds by applying Lemma 6 for each duration term.

   For the cases where ϕ^1 or ϕ^2 contain exclusively inequalities with duration terms, the proof is similar to this case.

- **Inductive cases**: For all formulas ψ_1 and ψ_2, Property 1 holds.

1. case ϕ^1 has inequalities:

   (a) containing temporal operators:

   Since ϕ^1 is a formula in DNF_3 containing temporal operators and inequalities of the form

   $$\left(W^1 \wedge Z^1\, U\, \phi_{\not<}\right) \vee \left(\neg W^1 \wedge R^1\, U\, \phi_{\not<}\right) \vee\ \cdots$$
   $$\vee\, (W^n \wedge Z^n\, U\, \phi_{\not<}) \vee (\neg W^n \wedge R^n\, U\, \phi_{\not<})$$

   containing an inequality formula W^i of the form $T^1 < T^2 \wedge \cdots \wedge T^j < T^{j+1}$, and two RMTL-$\int_3$ formulas Z^i,R^i in DNF_3 free of inequalities before an until operator occur (i.e., the new until operators can contain inequalities). Z^i and R^i are of the form

   $$\left(S_1^1\ U\ S_2^1 \wedge \cdots \wedge S_m^1\ U\ S_m^1\right) \vee \cdots \vee \left(S_1^n\ U\ S_2^n \wedge \cdots \wedge S_m^n\ U\ S_m^n\right),$$

where S is a RMTL-$\int_3$ formula in DNF_3.
From the inductive hypothesis, we have that any sub-formula $S_i^j \; U \; S_i^j$ has an equivalent formula where temporal operators are free of inequalities and this formula is a RMTL-$\int_3$ formula in DNF_3. Therefore, the formula may contain inequalities inner the until operator

$$Z^i \, U \, \phi_{\not<}.$$

Since there is no propositions, the replacement of the sub-formulas is straightforward. Applying the axiom A4, we have

$$\begin{aligned} F := \left(W^1 \wedge Z^1\right) \vee \left(\neg W^1 \wedge R^1\right) \vee \; \cdots \\ \vee \left(W^n \wedge Z^n\right) \vee \left(\neg W^n \wedge R^n\right) \end{aligned}$$

and

$$\begin{aligned} \left(W^1 \wedge F \, U \, \phi_{\not<}\right) \vee \left(\neg W^1 \wedge F^* \, U \, \phi_{\not<}\right) \vee \; \cdots \\ \vee \left(W^n \wedge F \, U \, \phi_{\not<}\right) \vee \left(\neg W^n \wedge F^* \, U \, \phi_{\not<}\right). \end{aligned}$$

Therefore, given that a conjunction/disjunction of a DNF_3 formula with other non-DNF_3 formula is a formula in DNF_3. Hence, the Property 1 holds.

(b) proposition and temporal operator free: The proof follows by the application of axiom A4.

2. case ϕ^2 has inequalities. This case is similar to previous one, but know using axiom A5 instead of A4.
3. case ϕ^1 and ϕ^2 have inequalities. The proof follows in a similar way to the previous two cases.

□

Let us now recall the Lemma 6.

Lemma 6. *Let ϕ be a formula in RMTL-$\int_3$, and η_x, η two terms, and consider the formula $\eta \sim \int^{\eta_x} \phi$. Then, there exists an equivalent formula where any duration term is free of inequalities, or only contains equalities of the form $x = \int^{\eta} \phi$.*

Proof of Lemma 6. By induction over the structure of the formula ϕ and the structure of the term η_x.

- **Base cases**:

1. ϕ does not contain any inequality:
   (a) η_x does not contain either logic variables or duration terms: the proof is straightforward.
   (b) η_x does not contain duration terms: the proof is straightforward since η_x is a rigid term.
2. ϕ contains inequalities without until operators, and η_x does not contain either logic variables or duration terms: Since there exists no terms that admit sub-formulas, any atom $X_{i,j}$ of a formula ϕ of the form

$$(X_{1,1} \wedge \cdots \wedge X_{1,m}) \vee \cdots \vee (X_{n,1} \wedge \cdots \wedge X_{n,m}),$$

   where $0 < i \leq n$ and $0 < j \leq m$, can only be a relation formula or a proposition. From the Axiom 7, we have

$$\eta \sim \int^{\eta_x} (X_{1,1} \wedge \cdots \wedge X_{1,m}) + \int^{\eta_x} (\cdots + (X_{n,1} \wedge \cdots \wedge X_{n,m})) - \int^{\eta_x} ((X_{1,1} \wedge \cdots \wedge X_{1,m}) \wedge \cdots \vee (X_{n,1} \wedge \cdots \wedge X_{n,m})).$$

   Continuing applying Axiom 7 until no disjunctions are left, we have a formula where the duration terms may only contain conjunctions of relation formulas and propositions. Then, we have

$$\eta \sim \int^{\eta_x} (X_{1,1} \wedge \cdots \wedge X_{1,m}) + \cdots + \int^{\eta_x} (X_{n,1} \wedge \cdots \wedge X_{n,m}) - \left(\int^{\eta_x} ((X_{1,1} \wedge \cdots \wedge X_{1,m}) \wedge (X_{n,1} \wedge \cdots \wedge X_{n,m})) + \ldots \right)$$

   Replacing each duration term by a logic variable, we get

$$\eta \sim (y_{1,1} + \cdots + y_{m,1}) - (y_{1,2} + \ldots).$$

   Now, applying Axiom 6 for each resulting duration term, we obtain a formula where inequalities are free of occurrences of the duration term. The resulting formula is of the form

$$f_{d_{(n)}}\left((y_{1,1}, \ldots, y_{m,1},),\ f^{\diamond}_{s_{(\phi^n_1)}},\ (m,1)\right) \wedge f_{d_{(n)}}\left(y_{1,2},\ f^{\diamond}_{s_{(\phi^n_2)}},\ (1,2)\right) \wedge \cdots$$

   Hence, Property 2 holds.
3. ϕ contains inequalities with until formulas and η_x does not contain either logic variables or duration terms: The proof structure is similar to the previous case, and then follows from Lemma 5.

- **Inductive cases**: For all ψ, r such that $\eta \sim \int^r \psi$, the Property 2 is true.

1. ϕ contains duration terms and η_x is a rigid term containing only constants or one logic variable:

   We assume ϕ is of the form $(X_{1,1} \wedge \cdots \wedge X_{1,m}) \vee \cdots \vee (X_{n,1} \wedge \cdots \wedge X_{n,m})$, where any atom $X_{i,j}$, $0 < i \leq n$ and $0 < j \leq m$ may contain a duration term. Applying the inductive hypothesis for each atom $X_{i,j}$ we have that

   $$E := \eta^* \sim y_{1,1} \times \alpha_{1,1} + \cdots + y_{n,m} \times \alpha_{n,m}$$

   and

   $$E \wedge D_n,$$

   which is a conjuntion of an inequality bounded by η^* and a disjunctive formula D_n containing several equalities of the form

   $$W^{1,1} \rightarrow y_{1,1} = \int^{\eta_x} Z^{1,1} \wedge \neg W^{1,1} \rightarrow y_{1,1} = 0 \wedge \ \cdots$$
   $$\wedge \ W^{n,m} \rightarrow y_{n,m} = \int^{\eta_x} Z^{n,m} \wedge \neg W^{n,m} \rightarrow y_{n,m} = 0,$$

   where W atoms are conjunctions of inequalities. Simplifying we get the formula

   $$\left(E \wedge W^{1,1} \wedge y_{1,1} = \int^{\eta_x} Z^{1,1}\right) \vee \left(E \wedge \neg W^{1,1} \wedge y_{1,1} = 0\right) \vee \cdots .$$

   Again applying Axiom 7 for the overall formula $\eta \sim \int^{\eta_x} \phi$ such that there is no disjunctions over it, we have

   $$\eta \sim \int^{\eta_x} \left(\left(E \wedge W^{1,1} \wedge y_{1,1} = \int^{\eta_x} Z^{1,1}\right) \vee \left(E \wedge \neg W^{1,1} \wedge y_{1,1} = 0\right) \wedge \cdots \wedge X_{1,m}\right) + \cdots .$$

   By applying Axiom 6, we get E and W free of duration terms. Hence, Property 2 holds.

2. ϕ does not contains inequalities and η_x contains duration terms:

   We assume η_x of the form $x_1 \times \alpha_1 + \cdots + x_n \times \alpha_n$ where α_i is replaced by a expression of the form $y_{i,1} \times \alpha_{i,1} + \cdots + y_{i,n} \times \alpha_{i,m}$ and so on replacing them until no logic variables are remaining. Then, we could simplify it by simply replacing the whole expression with a fresh logic variable using the Axiom 3. Then, we proceed with the same steps of the inductive case 1. Hence, Property 2 holds.

□

D.1 Soundness proofs for axioms

Let us now recall the introduced axioms.

A 1. $\eta_1 \circ \min_x \phi < \eta_2 \iff (\forall y\ y < x \to \neg\phi[y/x]) \wedge \eta_1 \circ x < \eta_2 \wedge \phi.$

A 2. $\eta_1 \circ \max_x \phi < \eta_2 \iff (\forall y\ y > x \to \neg\phi[y/x]) \wedge \eta_1 \circ x < \eta_2 \wedge \phi.$

A 3. $\int^{\eta_3} \phi_1 \circ \eta_1 \sim \eta_2 \iff x = \eta_3 \wedge \int^{x} \phi_1 \circ \eta_1 \sim \eta_2$

A 4. $\phi_1 \vee (\phi_r \wedge \phi_2)\ U\ \phi_3 \iff (\phi_r \to \phi_1 \vee \phi_2\, U\, \phi_3) \wedge (\neg\phi_r \to \phi_1\, U\, \phi_3)$

A 5. $\phi_1\, U\ (\phi_r \wedge \phi_2) \vee \phi_3 \iff (\phi_r \to \phi_1\, U\, \phi_2 \vee \phi_3) \wedge (\neg\phi_r \to \phi_1\, U\, \phi_3)$

A 6. $\int^{r} \phi_r \wedge \phi \sim \eta \iff \left(\phi_r \wedge \int^{r} \phi \sim \eta\right) \vee \left(\neg\phi_r \wedge \int^{r} \phi = 0\right)$

A 7. $\square \int^{\eta} \phi_1 \vee \phi_2 = \int^{\eta} \phi_1 + \int^{\eta} \phi_2 - \int^{\eta} \phi_1 \wedge \phi_2$

We have to prove the soundness of each one of the axioms, which means checking the validity of each axiom. The soundness proof of A3 is straightforward since it only replaces the term with a fresh variable. The soundness proof of A7 follows immediately from the semantics.

Lemma 7. *The axiom A4 is sound.*

Proof. The proof follows directly from the definition of the semantic interpretation of RMTL-$\int_3$ formulas. □

Lemma 8. *The axiom A5 is sound.*

Proof. The proof follows directly from the definition of the semantic interpretation of RMTL-$\int_3$ formulas. □

D.2 Application Examples

Example 22 (Duration term example)**.** *It illustrates for a specific case how simplification is done.*

1. $x < \int^{x+1} (P \wedge x < 10)$
   {*replace duration term by y*}
2. $x < y \wedge 0 \leq y \leq x + 1$
   {*apply weaker inequality for* $P \wedge x < 10$ }

3. $x < y \wedge 0 \leq y \leq x + 1 \wedge \Big((x < 10) \rightarrow \Big(0 \leq \int^{x+1} P \leq x + 1\Big)\Big) \wedge$
   $(\neg(x < 10) \rightarrow y = 0)$
   $\{$*replace new duration term by* $z\}$

4. $x < y \wedge 0 \leq y \leq x + 1 \wedge$
   $((x < 10) \rightarrow (0 \leq z \leq x + 1)) \wedge$
   $\neg(x < 10) \rightarrow y = 0$
   $\{$*apply CAD* $\}$

5. $y = 0 \wedge (z = 0 \vee (0 \leq z \leq x + 1))) \vee\ (0 < y \leq x + 1 \wedge 0 \leq z \leq x + 1)$ *for* $x \in [-1, 0[$,
   and $(x < y \leq x + 1 \wedge 0 \leq z \leq x + 1)$ *for* $x \in [0, 10[$
   $\{$*replace* y *and* z *by* $\int^{x+1} P$ $\}$

6. $\int^{x+1} P = 0 \vee 0 < \int^{x+1} P \leq x + 1$ *for* $x \in [-1, 0[$,
   $x < \int^{x+1} P \leq x + 1$ *for* $x \in [0, 10[$, *and*
   ff *otherwise*
   $\{$*simplify* $\int^{x+1} P \leq x + 1$ $\}$

7. $0 \leq \int^{x+1} P$ *for* $x \in [-1, 0[$,
   $x < \int^{x+1} P$ *for* $x \in [0, 10[$, *and*
   ff *otherwise*

Now, we have that $\forall x, x < \int^{x+1} (P \wedge x < 10)$ *is false, and* $\exists x, x < \int^{x+1} (P \wedge x < 10)$ *is true, since there is a value* $x = -1$ *where* $\int^{x+1} P = 0$.

After simplifying $\forall x, (0 \leq x < 10) \rightarrow x < \int^{x+1} (P \wedge x < 10)$, *we have* $\forall x, (0 \leq x < 10) \rightarrow x < \int^{x+1} P$.

Bibliography

Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Ondřej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding trace matching with free variables to aspectj. *SIGPLAN Not.*, 40(10):345–364, October 2005. ISSN 0362-1340. doi: 10.1145/1103845.1094839. URL http://doi.acm.org/10.1145/1103845.1094839.

José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simão Melo de Sousa. *Rigorous Software Development - An Introduction to Program Verification*. Undergraduate Topics in Computer Science. Springer, 2011. ISBN 978-0-85729-017-5. doi: 10.1007/978-0-85729-018-2. URL https://doi.org/10.1007/978-0-85729-018-2.

Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987. doi: 10.1007/BF01782772. URL https://doi.org/10.1007/BF01782772.

R. Alur and T.A. Henzinger. Logics and models of real time: A survey. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 74–106, London, UK, UK, 1992a. Springer-Verlag. ISBN 3-540-55564-1. URL http://dl.acm.org/citation.cfm?id=648143.749966.

Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi: 10.1016/0304-3975(94)90010-8. URL https://doi.org/10.1016/0304-3975(94)90010-8.

Rajeev Alur and Thomas A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*, pages 177–186, 1992b. doi: 10.1109/SFCS.1992.267774. URL https://doi.org/10.1109/SFCS.1992.267774.

Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993. doi: 10.1006/inco.1993.1025. URL https://doi.org/10.1006/inco.1993.1025.

Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994. doi: 10.1145/174644.174651. URL http://doi.acm.org/10.1145/174644.174651.

Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993. doi: 10.1006/inco.1993.1024. URL https://doi.org/10.1006/inco.1993.1024.

Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, January 1996. ISSN 0004-5411. doi: 10.1145/227595.227602. URL http://doi.acm.org/10.1145/227595.227602.

Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999. doi: 10.1016/S0304-3975(97)00173-4. URL https://doi.org/10.1016/S0304-3975(97)00173-4.

Miriam C. Bergue Alves, Doron Drusinsky, J. Bret Michael, and Man-tak Shing. Formal validation and verification of space flight software using statechart-assertions and runtime execution monitoring. In *6th International Conference on System of Systems Engineering, SoSE 2011, Albuquerque, New Mexico, USA, June 27-30, 2011*, pages 155–160, 2011. doi: 10.1109/SYSOSE.2011.5966590. URL https://doi.org/10.1109/SYSOSE.2011.5966590.

Björn Andersson and Jan Jonsson. Preemptive multiprocessor scheduling anomalies. In *16th International Parallel and Distributed Processing Symposium (IPDPS 2002), 15-19 April 2002, Fort Lauderdale, FL, USA, CD-ROM/Abstracts Proceedings*, 2002. doi: 10.1109/IPDPS.2002.1015483. URL https://doi.org/10.1109/IPDPS.2002.1015483.

June Andronick, Corey Lewis, Daniel Matichuk, Carroll Morgan, and Christine Rizkallah. Proof of OS scheduling behavior in the presence of interrupt-induced concurrency. In *Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, pages 52–68, 2016. doi: 10.1007/978-3-319-43144-4_4. URL https://doi.org/10.1007/978-3-319-43144-4_4.

Krzysztof R. Apt, Frank S. de Boer, and Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. Texts in Computer Science. Springer, 2009. ISBN 978-1-84882-744-8. doi: 10.1007/978-1-84882-745-5. URL https://doi.org/10.1007/978-1-84882-745-5.

Neil C. Audsley, Alan Burns, Robert I. Davis, Ken Tindell, and Andy J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*,

8(2-3):173–198, 1995. doi: 10.1007/BF01094342. URL https://doi.org/10.1007/BF01094342.

Mikhail Auguston and Mark B. Trakhtenbrot. Synthesis of monitors for real-time analysis of reactive systems. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, pages 72–86, 2008. doi: 10.1007/978-3-540-78127-1_5. URL https://doi.org/10.1007/978-3-540-78127-1_5.

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008. ISBN 026202649X, 9780262026499.

John Barnes. Rationale for ada 2012: Contracts and aspects. Technical report, Caversham, UK, 2012.

Julie Baro, Frédéric Boniol, Mikel Cordovilla, Eric Noulard, and Claire Pagetti. Off-line (optimal) multiprocessor scheduling of dependent periodic tasks. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1815–1820, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0857-1. doi: 10.1145/2245276.2232071. URL http://doi.acm.org/10.1145/2245276.2232071.

Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The smt-lib standard version 2.6. 2010.

Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, 2015. Available at `www.SMT-LIB.org`.

Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. *Rule-Based Runtime Verification*, pages 44–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004a. ISBN 978-3-540-24622-0. doi: 10.1007/978-3-540-24622-0_5. URL http://dx.doi.org/10.1007/978-3-540-24622-0_5.

Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In Bernhard Steffen and Giorgio Levi, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *Lecture Notes in Computer Science*, pages 277–306. Springer Berlin / Heidelberg, 2004b. ISBN 978-3-540-20803-7.

Howard Barringer, David Rydeheard, and Klaus Havelund. *Rule Systems for Run-Time Monitoring: From Eagle to RuleR*, pages 111–125. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-77395-5. doi: 10.1007/978-3-540-77395-5_10. URL http://dx.doi.org/10.1007/978-3-540-77395-5_10.

Howard Barringer, David Rydeheard, and Klaus Havelund. Rule systems for run-time monitoring. *J. Log. and Comput.*, 20(3):675–706, June 2010. ISSN 0955-792X. doi: 10.1093/logcom/exn076. URL http://dx.doi.org/10.1093/logcom/exn076.

Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 3540330984.

Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, September 2011. ISSN 1049-331X. doi: 10.1145/2000799.2000800. URL http://doi.acm.org/10.1145/2000799.2000800.

Gerd Behrmann, Alexandre David, Kim G. Larsen, John Hakansson, Paul Petterson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *Proceedings of the 3rd international conference on the Quantitative Evaluation of Systems*, QEST '06, pages 125–126, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2665-9. doi: 10.1109/QEST.2006.59. URL http://dx.doi.org/10.1109/QEST.2006.59.

Patrick Blackburn and Jerry Seligman. Hybrid languages. *Journal of Logic, Language and Information*, 4(3):251–272, Sep 1995. ISSN 1572-9583. doi: 10.1007/BF01049415. URL https://doi.org/10.1007/BF01049415.

Patrick Blackburn and Miroslava Tzakova. Hybrid languages and temporal logic. *Logic Journal of the IGPL*, 7(1):27–54, 1999. doi: 10.1093/jigpal/7.1.27. URL https://doi.org/10.1093/jigpal/7.1.27.

Eric Bodden. A lightweight LTL runtime verification tool for Java. In *Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004, October 24-28, 2004, Vancouver, BC, Canada*, pages 306–307. ACM, October 2004. URL http://www.bodden.de/pubs/bodden04lightweight.pdf. ACM Student Research Competition.

Borzoo Bonakdarpour, Samaneh Navabpour, and Sebastian Fischmeister. Sampling-based runtime verification. In *Proceedings of the 17th international conference on Formal methods*, FM'11, pages 88–102, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-21436-3. URL http://dl.acm.org/citation.cfm?id=2021296.2021308.

Borzoo Bonakdarpour, Samaneh Navabpour, and Sebastian Fischmeister. Time-triggered runtime verification. *Formal Methods in System Design*, 43(1):29–60, Aug 2013. ISSN 1572-8102. doi: 10.1007/s10703-012-0182-0. URL https://doi.org/10.1007/s10703-012-0182-0.

Patricia Bouyer. Model-checking timed temporal logics. *Electron. Notes Theor. Comput. Sci.*, 231:323–341, March 2009. ISSN 1571-0661. doi: 10.1016/j.entcs.2009.02.044. URL http://dx.doi.org/10.1016/j.entcs.2009.02.044.

Patricia Bouyer, Kim Guldstrand Larsen, and Nicolas Markey. Model checking one-clock priced timed automata. *Logical Methods in Computer Science*, 4(2), 2008a. doi: 10.2168/LMCS-4(2:9)2008. URL http://dx.doi.org/10.2168/LMCS-4(2:9)2008.

Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. On expressiveness and complexity in real-time model checking. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part II*, ICALP '08, pages 124–135, Berlin, Heidelberg, 2008b. Springer-Verlag. ISBN 978-3-540-70582-6. doi: 10.1007/978-3-540-70583-3_11. URL http://dx.doi.org/10.1007/978-3-540-70583-3_11.

Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. *Inf. Comput.*, 208(2):97–116, 2010. ISSN 0890-5401. doi: 10.1016/j.ic.2009.10.004. URL http://dx.doi.org/10.1016/j.ic.2009.10.004.

Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. Addison-Wesley Educational Publishers Inc, USA, 4th edition, 2009. ISBN 0321417453, 9780321417459.

Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems, Second Edition*. Springer, 2008. ISBN 978-0-387-33332-8. doi: 10.1007/978-0-387-68612-7. URL https://doi.org/10.1007/978-0-387-68612-7.

Franck Cassez and Kim Guldstrand Larsen. The impressive power of stopwatches. In *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, pages 138–152, 2000. doi: 10.1007/3-540-44618-4_12. URL https://doi.org/10.1007/3-540-44618-4_12.

Felipe Cerqueira, Felix Stutz, and Björn B. Brandenburg. PROSA: A case for readable mechanized schedulability analysis. In *28th Euromicro Conference on Real-Time Systems, ECRTS 2016, Toulouse, France, July 5-8, 2016*, pages 273–284, 2016. doi: 10.1109/ECRTS.2016.28. URL https://doi.org/10.1109/ECRTS.2016.28.

Zhou Chaochen, Anders P. Ravn, and Michael R. Hansen. An extended duration calculus for hybrid real-time systems. In *Hybrid Systems*, pages 36–59, London, UK, UK, 1993. Springer-Verlag. ISBN 3-540-57318-6. URL http://dl.acm.org/citation.cfm?id=646874.709980.

Feng Chen, Traian Florin Serbanuta, and Grigore Rosu. jpredictor: a predictive runtime analysis tool for java. In *Proceedings of the 30th international conference on Software*

engineering, ICSE '08, pages 221–230, New York, USA, 2008. ACM. ISBN 978-1-60558-079-1. doi: 10.1145/1368088.1368119. URL http://doi.acm.org/10.1145/1368088.1368119.

Alonzo Church. *The Calculi of Lambda-conversion*. Annals of mathematics studies. Princeton University Press, 1941. ISBN 9780691083940.

Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-03270-8.

George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition: a synopsis. *ACM SIGSAM Bulletin*, 10(1):10–12, 1976. doi: 10.1145/1093390.1093393. URL http://doi.acm.org/10.1145/1093390.1093393.

M. Coombes, O. McAree, W. H. Chen, and P. Render. Development of an autopilot system for rapid prototyping of high level control algorithms. In *Proceedings of 2012 UKACC CONTROL*, pages 292–297, Sept 2012. doi: 10.1109/CONTROL.2012.6334645.

Marcelo d'Amorim and Klaus Havelund. Event-based runtime verification of java programs. In *Proceedings of the third international workshop on Dynamic analysis*, WODA '05, pages 1–7, New York, USA, 2005. ACM. ISBN 1-59593-126-0. doi: 10.1145/1082983.1083249. URL http://doi.acm.org/10.1145/1082983.1083249.

Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. LOLA: runtime monitoring of synchronous systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*, pages 166–174, 2005. doi: 10.1109/TIME.2005.26. URL https://doi.org/10.1109/TIME.2005.26.

Rowan Davies. A temporal logic approach to binding-time analysis. *J. ACM*, 64(1):1:1–1:45, March 2017. ISSN 0004-5411. doi: 10.1145/3011069. URL http://doi.acm.org/10.1145/3011069.

Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, October 2011. ISSN 0360-0300. doi: 10.1145/1978802.1978814. URL http://doi.acm.org/10.1145/1978802.1978814.

André De Matos Pedro. *rtmlib* Monitoring Library, 2016. Available at https://anmaped.github.io/rtmlib/doc/, version 0.1-alpha.

André De Matos Pedro. *rmtld3synth* Synthesis Tool, 2018. Available at https://github.com/anmaped/rmtld3synth/, version 0.3-alpha2.

André De Matos Pedro, David Pereira, Luís Miguel Pinho, and Jorge Sousa Pinto. Towards a runtime verification framework for the ada programming language. In *Reliable Software Technologies - Ada-Europe 2014, 19th Ada-Europe International Conference on Reliable Software Technologies, Paris, France, June 23-27, 2014. Proceedings*, pages 58–73, 2014a. doi: 10.1007/978-3-319-08311-7_6. URL https://doi.org/10.1007/978-3-319-08311-7_6.

André De Matos Pedro, David Pereira, Luís Miguel Pinho, and Jorge Sousa Pinto. A compositional monitoring framework for hard real-time systems. In *Proceedings of the 6th International Symposium on NASA Formal Methods - Volume 8430*, pages 16–30, New York, NY, USA, 2014b. Springer-Verlag New York, Inc. ISBN 978-3-319-06199-3. doi: 10.1007/978-3-319-06200-6_2. URL http://dx.doi.org/10.1007/978-3-319-06200-6_2.

André De Matos Pedro, David Pereira, Luís Miguel Pinho, and Jorge Sousa Pinto. Monitoring for a decidable fragment of mtl-$\int$. In *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, pages 169–184, 2015a. doi: 10.1007/978-3-319-23820-3_11. URL https://doi.org/10.1007/978-3-319-23820-3_11.

André De Matos Pedro, David Pereira, Luís Miguel Pinho, and Jorge Sousa Pinto. Logic-based Schedulability Analysis for Compositional Hard Real-time Embedded Systems. *SIGBED Rev.*, 12(1):56–64, March 2015b. ISSN 1551-3688. doi: 10.1145/2752801.2752808. URL http://doi.acm.org/10.1145/2752801.2752808.

André De Matos Pedro, David Pereira, Luis Miguel Pinho, and Jorge Sousa Pinto. SMT-based schedulability analysis using RMTL-$\int$. *CRTS 2016*, page 31, 2016.

André De Matos Pedro, Jorge Sousa Pinto, David Pereira, and Luís Miguel Pinho. Runtime verification of autopilot systems using a fragment of MTL-$\int$. *International Journal on Software Tools for Technology Transfer*, Aug 2017. ISSN 1433-2787. doi: 10.1007/s10009-017-0470-5. URL https://doi.org/10.1007/s10009-017-0470-5.

Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008. doi: 10.1007/978-3-540-78800-3_24. URL https://doi.org/10.1007/978-3-540-78800-3_24.

Doron Drusinsky. The temporal rover and the atg rover. In *Proceedings of the 7th International SPIN Workshop on SPIN Model Checking and Software Verification*,

pages 323–330, London, UK, 2000. Springer-Verlag. ISBN 3-540-41030-9. URL http://dl.acm.org/citation.cfm?id=645880.672089.

E. Allen Emerson. Handbook of theoretical computer science (vol. b). chapter Temporal and Modal Logic, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990. ISBN 0-444-88074-7. URL http://dl.acm.org/citation.cfm?id=114891.114907.

Yliès Falcone. You should better enforce than verify. In *Proceedings of the First international conference on Runtime verification*, RV'10, pages 89–105, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16611-3, 978-3-642-16611-2.

Elena Fersman, Pavel Krcal, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation*, 205(8):1149–1172, August 2007. ISSN 0890-5401. doi: 10.1016/j.ic.2007.01.009. URL http://dx.doi.org/10.1016/j.ic.2007.01.009.

C. J. Fidge. Real-time schedulability tests for preemptive multitasking. *Real-Time Syst.*, 14(1):61–93, January 1998. ISSN 0922-6443.

Sebastian Fischmeister and Yanmeng Ba. Sampling-based program execution monitoring. In *Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems*, LCTES '10, pages 133–142, New York, USA, 2010. ACM. ISBN 978-1-60558-953-4. doi: 10.1145/1755888.1755908. URL http://doi.acm.org/10.1145/1755888.1755908.

Melvin Fitting. *First-Order Logic and Automated Theorem Proving, Second Edition*. Graduate Texts in Computer Science. Springer, 1996. ISBN 978-1-4612-7515-2. doi: 10.1007/978-1-4612-2360-3. URL https://doi.org/10.1007/978-1-4612-2360-3.

Carlo A. Furia and Matteo Rossi. On the expressiveness of MTL variants over dense time. In *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings*, pages 163–178, 2007. doi: 10.1007/978-3-540-75454-1_13. URL https://doi.org/10.1007/978-3-540-75454-1_13.

Dov M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Temporal Logic in Specification, Altrincham, UK, April 8-10, 1987, Proceedings*, pages 409–448, 1987. doi: 10.1007/3-540-51803-7_36. URL https://doi.org/10.1007/3-540-51803-7_36.

Simon Goldsmith, Robert O'Callahan, and Alexander Aiken. Relational queries over program traces. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2005,*

October 16-20, 2005, San Diego, CA, USA, pages 385–402, 2005. doi: 10.1145/1094811.1094841. URL http://doi.acm.org/10.1145/1094811.1094841.

Alwyn Goodloe and Lee Pike. Monitoring distributed real-time systems: A survey and future directions. Technical Report NASA/CR-2010-216724, NASA Langley Research Center, July 2010.

Joël Goossens, Emmanuel Grolleau, and Liliana Cucu-Grosjean. Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. *Real-Time Syst.*, 52(6):808–832, November 2016. ISSN 0922-6443. doi: 10.1007/s11241-016-9256-1. URL http://dx.doi.org/10.1007/s11241-016-9256-1.

Russell A. Gordon. *The Integrals of Lebesgue, Denjoy, Perron, and Henstock*. Graduate studies in mathematics. American Mathematical Soc., 1994. ISBN 9780821872222.

Dick Hamlet. *Composing Software Components: A Software-testing Perspective*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 1441971475, 9781441971470.

David Harel and Amnon Naamad. The statemate semantics of statecharts. *ACM Trans. Softw. Eng. Methodol.*, 5(4):293–333, October 1996. ISSN 1049-331X. doi: 10.1145/235321.235322. URL http://doi.acm.org/10.1145/235321.235322.

David Harel, Jerzy Tiuryn, and Dexter Kozen. *Dynamic Logic*. MIT Press, Cambridge, MA, USA, 2000. ISBN 0262082896.

John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, New York, NY, USA, 1st edition, 2009. ISBN 0521899575, 9780521899574.

Klaus Havelund. Runtime verification of C programs. In *Proceedings of the 20th IFIP TC 6/WG 6.1 international conference on Testing of Software and Communicating Systems: 8th International Workshop*, TestCom '08 / FATES '08, pages 7–22, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-68514-2. doi: 10.1007/978-3-540-68524-1_3. URL http://dx.doi.org/10.1007/978-3-540-68524-1_3.

Klaus Havelund and Grigore Rosu. Monitoring java programs with java pathexplorer. *Electr. Notes Theor. Comput. Sci.*, 55(2):200–217, 2001. doi: 10.1016/S1571-0661(04)00253-1. URL https://doi.org/10.1016/S1571-0661(04)00253-1.

Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS '02, pages 342–356, London, UK, 2002. Springer-Verlag. ISBN 3-540-43419-4. URL http://dl.acm.org/citation.cfm?id=646486.694486.

Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, pages 545–558, 1992. doi: 10.1007/3-540-55719-9_103. URL https://doi.org/10.1007/3-540-55719-9_103.

J. Roger Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators: An Introduction*. Cambridge University Press, New York, NY, USA, 2nd edition, 2008. ISBN 0521898854, 9780521898850.

Yoram Hirshfeld and Alexander Rabinovich. Logics for real time: Decidability and complexity. *Fundam. Inf.*, 62(1):1–28, January 2004. ISSN 0169-2968. URL http://dl.acm.org/citation.cfm?id=1227039.1227041.

Gabriel M. Hoffmann, Haomiao Huang, Steven L. Wasl, and Er Claire J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proc. of the AIAA Guidance, Navigation, and Control Conference. Vol. 2.*, 2007.

Paul Hunter, Joël Ouaknine, and James Worrell. Expressive completeness for metric temporal logic. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 349–357, 2013. doi: 10.1109/LICS.2013.41. URL https://doi.org/10.1109/LICS.2013.41.

Dongyun Jin, Patrick O'Neil Meredith, Choonghwan Lee, and Grigore Roşu. Javamop: Efficient parametric runtime monitoring framework. In *Proceeding of the 34th International Conference on Software Engineering (ICSE'12)*. IEEE, 2012. to appear.

Dejan Jovanović and Leonardo de Moura. Solving non-linear arithmetic. *ACM Commun. Comput. Algebra*, 46(3/4):104–105, January 2013. ISSN 1932-2240. doi: 10.1145/2429135.2429155. URL http://doi.acm.org/10.1145/2429135.2429155.

Simon J. Julier and Jeffrey K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004. doi: 10.1109/JPROC.2003.823141. URL https://doi.org/10.1109/JPROC.2003.823141.

Moonzoo Kim, Mahesh Viswanathan, Sampath Kannan, Insup Lee, and Oleg Sokolsky. Java-mac: A run-time assurance approach for java programs. *Form. Methods Syst. Des.*, 24(2):129–155, March 2004. ISSN 0925-9856. doi: 10.1023/B:FORM.0000017719.43755.7c. URL http://dx.doi.org/10.1023/B:FORM.0000017719.43755.7c.

Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, October 1990. ISSN 0922-6443. doi: 10.1007/BF01995674. URL http://dx.doi.org/10.1007/BF01995674.

Pavel Krcal, Martin Stigge, and Wang Yi. Multi-processor schedulability analysis of preemptive real-time tasks with variable execution times. In *Proceedings of the 5th international conference on Formal modeling and analysis of timed systems*, FORMATS'07, pages 274–289, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-75453-9, 978-3-540-75453-4. URL http://dl.acm.org/citation.cfm?id=1779879.1779899.

Shankara Narayanan Krishna, Khushraj Madnani, and Paritosh K. Pandya. Metric temporal logic with counting. In *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 335–352, 2016. doi: 10.1007/978-3-662-49630-5_20. URL https://doi.org/10.1007/978-3-662-49630-5_20.

Yassine Lakhnech and Jozef Hooman. Metric temporal logic with durations. *Theor. Comput. Sci.*, 138(1):169–199, 1995. doi: 10.1016/0304-3975(94)00151-8. URL https://doi.org/10.1016/0304-3975(94)00151-8.

François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Temporal logic with forgettable past. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 383–392, 2002. doi: 10.1109/LICS.2002.1029846. URL https://doi.org/10.1109/LICS.2002.1029846.

Christopher League. Lambda calculi: A guide for computer scientists by chris hankin. *SIGACT News*, 31(1):8–13, March 2000. ISSN 0163-5700. doi: 10.1145/346048.568490. URL http://doi.acm.org/10.1145/346048.568490.

John P. Lehoczky, Lui Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989, Santa Monica, California, USA, December 1989*, pages 166–171, 1989. doi: 10.1109/REAL.1989.63567. URL https://doi.org/10.1109/REAL.1989.63567.

Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973. ISSN 0004-5411. doi: 10.1145/321738.321743. URL http://doi.acm.org/10.1145/321738.321743.

Hong Lu and A. Forin. Automatic processor customization for zero-overhead online software verification. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(10):1346 –1357, October 2008. ISSN 1063-8210.

David Makinson. *Sets, Logic and Maths for Computing, Second Edition*. Undergraduate Topics in Computer Science. Springer, 2012. ISBN 978-1-4471-2499-3. doi: 10.1007/978-1-4471-2500-6. URL https://doi.org/10.1007/978-1-4471-2500-6.

Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT*, pages 152–166, 2004.

Rajib Mall. *Real-Time Systems: Theory and Practice*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2009. ISBN 8131700690, 9788131700693.

Michael C. Martin, V. Benjamin Livshits, and Monica S. Lam. Finding application errors and security flaws using PQL: a program query language. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2005, October 16-20, 2005, San Diego, CA, USA*, pages 365–383, 2005. doi: 10.1145/1094811.1094840. URL http://doi.acm.org/10.1145/1094811.1094840.

Ramy Medhat, Borzoo Bonakdarpour, Deepak Kumar, and Sebastian Fischmeister. Runtime monitoring of cyber-physical systems under timing and memory constraints. *ACM Trans. Embed. Comput. Syst.*, 14(4):79:1–79:29, October 2015. ISSN 1539-9087. doi: 10.1145/2744196. URL http://doi.acm.org/10.1145/2744196.

Lorenz Meier, Dominik Honegger, and Marc Pollefeys. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 6235–6240, 2015. doi: 10.1109/ICRA.2015.7140074. URL https://doi.org/10.1109/ICRA.2015.7140074.

Patrick Meredith and Grigore Roşu. Runtime verification with the rv system. In *Proceedings of the First international conference on Runtime verification*, RV'10, pages 136–152, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16611-3, 978-3-642-16611-2. URL http://dl.acm.org/citation.cfm?id=1939399.1939413.

Patrick O'Neil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Roşu. An overview of the MOP runtime verification framework. *International Journal on Software Techniques for Technology Transfer*, 2011.

Patrick Moosbrugger, Kristin Y. Rozier, and Johann Schumann. R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. *Formal Methods in System Design*, 51(1):31–61, 2017. doi: 10.1007/s10703-017-0275-x. URL https://doi.org/10.1007/s10703-017-0275-x.

Mark W. Müller and Raffaello D'Andrea. Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pages 45–52, 2014. doi: 10.1109/ICRA.2014.6906588. URL https://doi.org/10.1109/ICRA.2014.6906588.

Samaneh Navabpour, Borzoo Bonakdarpour, and Sebastian Fischmeister. Time-triggered runtime verification of component-based multi-core systems. In *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, pages 153–168, 2015. doi: 10.1007/978-3-319-23820-3_10. URL https://doi.org/10.1007/978-3-319-23820-3_10.

G. Nelissen, D. Pereira, and L. M. Pinho. A novel run-time monitoring architecture for safe and efficient inline monitoring. In *Ada-Europe 2015*, pages 66–82, June 2015.

Dejan Nickovic and Nir Piterman. From mtl to deterministic timed automata. In *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, pages 152–167, 2010. doi: 10.1007/978-3-642-15297-9_13. URL https://doi.org/10.1007/978-3-642-15297-9_13.

Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of program analysis*. Springer, 1999. ISBN 978-3-540-65410-0. doi: 10.1007/978-3-662-03811-6. URL https://doi.org/10.1007/978-3-662-03811-6.

Christer Norström, Anders Wall, and Wang Yi. Timed automata as task models for event-driven systems. In *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, RTCSA '99, pages 182–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0306-3. URL http://dl.acm.org/citation.cfm?id=519167.828781.

Paritosh K. Pandya and Simoni S. Shah. Unambiguity in timed regular languages: Automata and logics. In *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, pages 168–182, 2010. doi: 10.1007/978-3-642-15297-9_14. URL https://doi.org/10.1007/978-3-642-15297-9_14.

Pawel Parys and Igor Walukiewicz. Weak alternating timed automata. In *Proceedings of the 36th Internatilonal Collogquium on Automata, Languages and Programming: Part II*, ICALP '09, pages 273–284, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-02929-5. doi: 10.1007/978-3-642-02930-1_23. URL http://dx.doi.org/10.1007/978-3-642-02930-1_23.

Rodolfo Pellizzoni, Patrick Meredith, Marco Caccamo, and Grigore Rosu. Hardware runtime monitoring for dependable cots-based real-time embedded systems. In *Proceedings of the 2008 Real-Time Systems Symposium*, RTSS '08, pages 481–491, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3477-0. doi: 10.1109/RTSS.2008.43. URL http://dx.doi.org/10.1109/RTSS.2008.43.

Lee Pike, Alwyn Goodloe, Robin Morisset, and Sebastian Niller. Copilot: A hard real-time runtime monitor. In *Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, pages 345–359, 2010. doi: 10.1007/978-3-642-16612-9_26. URL https://doi.org/10.1007/978-3-642-16612-9_26.

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Landry Nguena Timo. *Runtime Enforcement of Timed Properties*, pages 229–244. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-35632-2. doi: 10.1007/978-3-642-35632-2_23. URL https://doi.org/10.1007/978-3-642-35632-2_23.

André Platzer. Towards a hybrid dynamic logic for hybrid dynamic systems. *Electron. Notes Theor. Comput. Sci.*, 174(6):63–77, June 2007. ISSN 1571-0661. doi: 10.1016/j.entcs.2006.11.026. URL http://dx.doi.org/10.1016/j.entcs.2006.11.026.

André Platzer. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning*, 41(2):143–189, Aug 2008. ISSN 1573-0670. doi: 10.1007/s10817-008-9103-8. URL https://doi.org/10.1007/s10817-008-9103-8.

Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society. doi: 10.1109/SFCS.1977.32. URL http://dx.doi.org/10.1109/SFCS.1977.32.

Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti. Off-line mapping of multi-rate dependent task sets to many-core platforms. *Real-Time Syst.*, 51(5):526–565, September 2015. ISSN 0922-6443. doi: 10.1007/s11241-015-9232-1. URL http://dx.doi.org/10.1007/s11241-015-9232-1.

Mina Ranjbaran and Khashayar Khorasani. Fault recovery of an under-actuated quadrotor aerial vehicle. In *Proceedings of the 49th IEEE Conference on Decision and Control, CDC 2010, December 15-17, 2010, Atlanta, Georgia, USA*, pages 4385–4392, 2010. doi: 10.1109/CDC.2010.5718140. URL https://doi.org/10.1109/CDC.2010.5718140.

Didier Rémy. *Using, Understanding, and Unraveling the OCaml Language From Practice to Theory and Vice Versa*, pages 413–536. Springer Berlin Heidelberg, Berlin,

Heidelberg, 2002. ISBN 978-3-540-45699-5. doi: 10.1007/3-540-45699-6_9. URL http://dx.doi.org/10.1007/3-540-45699-6_9.

Usa Sammapun, Insup Lee, Oleg Sokolsky, and John Regehr. Statistical runtime checking of probabilistic properties. In *Proceedings of the 7th international conference on Runtime verification*, RV'07, pages 164–175, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-77394-0, 978-3-540-77394-8. URL http://dl.acm.org/citation.cfm?id=1785141.1785158.

Sriram Sankar and Manas Mandal. Concurrent runtime monitoring of formally specified programs. *Computer*, 26(3):32–41, March 1993. ISSN 0018-9162. doi: 10.1109/2.204684. URL http://dx.doi.org/10.1109/2.204684.

Lui Sha, Tarek Abdelzaher, Karl-Erik én, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Syst.*, 28(2-3):101–155, November 2004. ISSN 0922-6443.

Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003), 3-5 December 2003, Cancun, Mexico*, pages 2–13, 2003. doi: 10.1109/REAL.2003.1253249. URL https://doi.org/10.1109/REAL.2003.1253249.

Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embedded Comput. Syst.*, 7(3):30:1–30:39, 2008. doi: 10.1145/1347375.1347383. URL http://doi.acm.org/10.1145/1347375.1347383.

Oleg Sokolsky, Usa Sammapun, Insup Lee, and Jesung Kim. Run-time checking of dynamic properties. *Electron. Notes Theor. Comput. Sci.*, 144(4):91–108, May 2006. ISSN 1571-0661. doi: 10.1016/j.entcs.2006.02.006. URL http://dx.doi.org/10.1016/j.entcs.2006.02.006.

Deepak Souza and Pavithra Prabhakar. On the expressiveness of mtl in the pointwise and continuous semantics. *Int. J. Softw. Tools Technol. Transf.*, 9(1):1–4, February 2007. ISSN 1433-2779. doi: 10.1007/s10009-005-0214-9. URL http://dx.doi.org/10.1007/s10009-005-0214-9.

John A. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 21(10):10–19, 1988. doi: 10.1109/2.7053. URL https://doi.org/10.1109/2.7053.

Karl Johan Åström and Tore Hägglund. *Advanced PID Control*. ISA - The Instrumentation, Systems and Automation Society, 2006. ISBN 978-1-55617-942-6.

Alfred Tarski. *Introduction to Logic and to the Methodology of Deductive Sciences*. Dover Books on Mathematics Series. Dover Publications, 1995. ISBN 9780486284620.

The OCaml Development Team. Ocaml programming language, 2013. URL http://www.ocaml.org.

Julian Tschannen, Carlo A. Furia, Martin Nordio, and Bertrand Meyer. Usable verification of object-oriented programs by combining static and dynamic techniques. In *Software Engineering and Formal Methods - 9th International Conference, SEFM 2011, Montevideo, Uruguay, November 14-18, 2011. Proceedings*, pages 382–398, 2011. doi: 10.1007/978-3-642-24690-6_26. URL https://doi.org/10.1007/978-3-642-24690-6_26.

Karen Zee, Viktor Kuncak, Michael Taylor, and Martin Rinard. Runtime checking for program verification. In *Proceedings of the 7th international conference on Runtime verification*, RV'07, pages 202–213, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-77394-0, 978-3-540-77394-8. URL http://dl.acm.org/citation.cfm?id=1785141.1785161.

Haitao Zhu, Matthew B. Dwyer, and Steve Goddard. Predictable runtime monitoring. In *21st Euromicro Conference on Real-Time Systems, ECRTS 2009, Dublin, Ireland, July 1-3, 2009*, pages 173–183, 2009. doi: 10.1109/ECRTS.2009.23. URL https://doi.org/10.1109/ECRTS.2009.23.

Haitao Zhu, Steve Goddard, and Matthew B. Dwyer. Selecting server parameters for predictable runtime monitoring. In *16th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2010, Stockholm, Sweden, April 12-15, 2010*, pages 227–236, 2010. doi: 10.1109/RTAS.2010.18. URL https://doi.org/10.1109/RTAS.2010.18.